Teklemariam et al. EURASIP Journal on Wireless Communications and Networking 2013, 2013:177
http://jwcn.eurasipjournals.com/content/2013/1/177

EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

Facilitating the creation of IoT applications through conditional observations in CoAP

Girum Ketema Teklemariam, Jeroen Hoebeke*, Ingrid Moerman and Piet Demeester

Abstract

With the advent of IPv6, the world is getting ready to incorporate smart objects to the current Internet to realize the idea of Internet of Things. The biggest challenge faced is the resource constraint of the smart objects to directly utilize the existing standard protocols and applications. A number of initiatives are currently witnessed to resolve this situation. One of such initiatives is the introduction of Constrained Application Protocol. This protocol is developed to fit in the resource-constrained smart object with the ability to easily translate to the prominent representational state transfer implementation, hypertext transfer protocol (and vice versa). The protocol has several optional extensions, one of them being, resource observation. With resource observation, a client may ask a server to be notified every state change of the resource. However, in many applications, all state changes are not significant enough for the clients. Therefore, the client will have to decide whether to use a value sent by a server or not. This results in wastage of the already constrained resources (bandwidth, processing power,…). In this paper, we introduced an alternative to the normal resource observation function, named Conditional Observation, where clients tell the servers the criteria for notification. We evaluated the power consumption and number of packets transmitted between clients and servers by using different network sizes and number of servers. In all cases, we found out that the existing observe option results in excessive number of packets (most of them unimportant for the client) and higher power consumption. We also made an extensive theoretical evaluation of the two approaches which give consistent result with the results we got from experimentation.

Keywords: IoT; Conditional observation; Resource observation; REST; CoAP

1 Introduction

Remarkable advances in microelectromechanical systems (MEMS) have led to the creation of tiny but crucial embedded devices such as sensors and actuators. The wireless communication capability of these devices turns them into smart objects that can interact with the virtual world. Coupled with the explosive expansion of wireless and mobile technologies, there are very good reasons to consider these objects as corner stones of the future internet rather than mere add-ons to the current communication networks. The resulting Internet is now commonly referred to as the internet of things (IoT). However, the severe limitations of these smart objects in terms of memory, processing capacity, power, and bandwidth pose great challenges in realizing this. A typical smart object may have a few kilobytes of memory (random access memory (RAM) and read-only memory (ROM)), slow microcontrollers, and limited bandwidth (around 250 kbps). On top of this, most of the smart objects are battery operated and have limited lifetime. The protocols and applications that are widely used in the current internet are too heavy for such constrained devices to be applied directly. Several initiatives exist to alleviate these prevailing problems by proposing new lightweight protocols suitable for constrained devices and networks. The internet engineering task force (IETF) is the pioneer in producing standards and protocols that fit the strict requirements of such constrained environments by establishing working groups that address different aspects of the requirements of the constrained objects and networks.

The IPv6 for low power and lossy wireless personal area network (6LoWPAN) working group of IETF has produced standards that enable IPv6 to be used in the most constrained devices [1]. Montenegro et al. [2] introduce the 6LoWPAN adaptation layer which resides

* Correspondence: jeroen.hoebeke@intec.ugent.be
Department of Information Technology (INTEC), Ghent University – iMinds, Gaston Crommenlaan 8 Bus 201, Ghent 9050, Belgium

between the network and link layer and provides three basic services: IPv6 header compression, fragmentation, and mesh under routing support. These basic services ensure that constrained devices can talk to unmodified IPv6 hosts in the internet, and the other way around, while the 6LoWPAN adaption layer overcomes the differences in protocol design between these two worlds, necessitated by the constraints of the low power and lossy networks (LLNs). Further, current routing protocols and algorithms are not suitable for constrained environments for several reasons: high resource (memory, processing, and bandwidth) requirement, absence of uniform metric in LLNs and unreliability of intermediate routing nodes. The routing in low power and lossy networks (ROLL) working group is tasked with proposing routing solutions suitable for constrained networks and devices. The routing protocol for low-power and lossy networks (RPL) is a proposed standard by this working group [3].

Both IETF groups have realized the interconnectivity between tiny objects and the current internet in a standardized way. However, this connectivity is merely an enabler required to unlock all potential of the IoT in the form of novel applications and services. Web service technology made the success of the current internet. Now, it is expected that an embedded counterpart of web service technology is needed in order to exploit all great opportunities offered by the internet of things, since existing application layer protocols, such as hypertext transfer protocol (HTTP), SOAP, and XML are even heavier than the protocols defined in layers below. Therefore, the constrained RESTful environments (CoRE) working group was established to specifically work on the standardization of a framework for resource-oriented applications, allowing the realization of RESTful embedded web services in a similar way as traditional web services [4]. Their work resulted in the Constrained Application Protocol (CoAP), a specialized RESTful web transfer protocol for use with constrained networks and nodes. It uses the same RESTful principles as HTTP, but it is much lighter so that it can be run on constrained devices [4]. In addition, the group designed observe functionality in order to allow a device to publish a value or event to another device that has subscribed to be notified of changes in the resource representation [5].

CoAP, together with its observe functionality, provides the basis for the integration of constrained devices with the internet at the service level and the realization of embedded web services. However, in order to really facilitate IoT application design, additional CoAP-related functionalities are expected to appear. For instance, many applications can benefit from a lightweight solution for subscribing for very specific events. Ideally, this is built into the CoAP protocol as an extension, avoiding the need to implement such functionality on a per resource basis. This facilitates the realization of many sensor–actuator interactions which typically have the following pattern: if 'condition fulfilled' then 'take action'. The contribution of this paper is that we present an extension of the CoAP observe functionality that exactly facilitates realizing this behavior by including notification criteria to be specified along with observation request. This way, the server will not just send notifications whenever the state of a resource changes. It will first check if the change is significant enough for the client by comparing the new value with the notification criteria sent by the client. Only then, a notification will be sent. The design is compact, lightweight, and can be easily shared across all resources. Further, we are the first to implement such an extension to CoAP on constrained devices and to evaluate in detail the potential reduction in power consumption and number of packets transmitted that can be achieved, which is of great importance to constrained networks.

Section 2 of the paper first introduces the CoAP protocol, followed by the existing CoAP Observe option, its limitations, and possible approach to tackle these limitations. Related work will be discussed in section 3. In section 4, an alternative method, called conditional observation, is presented, and the approach is explained in great details. The next section discusses our implementation on constrained devices, followed by section 6 presenting a detailed experimental and mathematical evaluation. In section 7, we further illustrate some potential IoT applications that can benefit from our proposal. Finally, the paper draws conclusions and suggests future work.

2 The Constrained Application Protocol and observe

Representational state transfer (REST) uses mechanisms that are less memory and processing power intensive [6]. As a result, many systems are now becoming RESTful [4]. In this approach, data or resources that must be exchanged between client and server are encoded as representations of the resource. In addition, all states required to complete a request must be provided along with the request. The desired communication result is achieved by transferring the representations and the states between the client and the server using HTTP operations such as GET, PUT, POST, and DELETE [6]. However, today's web service technology is a poor match for the vast majority of constrained networks, machine-to-machine (M2M) applications and embedded devices because of their overhead and complexity. For applications that involve smart objects, such as industry automation, transport logistics, and building automation, an embedded alternative would be ideal since it is in line with current web services, facilitating the integration of objects into the Internet.

CoAP is a protocol proposed by the IETF CoRE working group allowing these RESTful web services to be

implemented on constrained objects. CoAP provides exactly the subset of HTTP methods (GET, PUT, POST, and DELETE) that is necessary to offer RESTful web services in a WSN-compatible manner [4]. This implies that a simple mapping between HTTP and CoAP can be realized (and vice versa) in a similar way that 6LoWPAN can be translated into IPv6 and the other way around. The main advantage is that CoAP has a much lower header overhead and parsing complexity than HTTP. It uses a 4-byte base binary header that may be followed by compact binary options and a payload. In addition, CoAP provides optional transport reliability, normally a core functionality of TCP, which is due to the resource constraints by nature not available in wireless sensor networks (WSNs). This is particularly useful, since CoAP is designed to be used in combination with UDP, which does not offer any reliability but is adequate for WSNs due to its low impact on resources. CoAP can run on top of 6LoWPAN networks, but also on top of proprietary networks that are connected to IPv6 internet. Figure 1 shows the CoAP message format as specified in version 13 of the draft [4]. The 4-bytes base header consists of the following fields: Version, Type, Token length, Code, and Message ID. The 2-bit Type field indicates whether the message is a confirmable, non-confirmable, acknowledgement, or reset message. The Code field indicates if the message carries a request (specifying the method: GET, PUT, POST, or DELETE), response (specifying the response code) or is empty. The base header may be followed by one or more optional fields: first of all, there is the optional Token field having a length between 0 and 8 bytes; next, a variable number of options can follow; and finally, if there is a payload, a Payload Marker and the Payload complete the message.

The format of a single CoAP option is shown in Figure 2. To be able to offer communication needs that cannot be satisfied by the base binary header alone, CoAP defines a number of options which can be included in a message. Each option instance in a message specifies the option number of the defined CoAP option. Instead of specifying the Option Number directly, the instances must appear in order of their Option Numbers and a delta encoding is used between them. The Option Length indicates the length of the Option Value in bytes, and the Option Value is the actual representation of the option (e.g., an unsigned integer, a code representation, etc.). If the delta value or length is larger than 12, 1 or 2 additional bytes are used to represent the delta or the length.

Since CoAP is recommended for M2M interaction, automatic resource discovery is made part of the protocol using the CoRE link format. A well-known URI, '/.well-known/core', is defined as an entry point for all links to resources hosted by a server [7]. Once the list of resources is identified, clients may send requests to find out specific values for the resources. As Figure 3 depicts, the client first requests the list of resources using GET, and the server replies with the list of resources it has. At a later time, the client requests for the current temperature value using another GET, to which the server replies with a response containing the temperature value of 23.5°C. All exchanges use the message format shown in Figures 1 and 2.

In addition to the main CoAP draft, a number of extensions have been proposed. One of those extensions is the observation of resources through the use of the Observe option. The Observe option may be used by clients interested to have up-to-date information about the state of a resource as stated in [5]. This draft specifies a simple protocol extension to CoAP that gives clients the ability to observe changes of a resource. It uses the well-known observer design pattern, where clients that are interested in the state of a resource register their interest with the server that hosts the resource by sending a CoAP request containing the Observe option. Once registered, clients will receive notifications - CoAP responses containing the Observe option - upon every state change of the resource. In addition, if the state of a resource does not change over time, the server will send a new notification latest after MAX-AGE of the resource expires. Since the CoAP option MAX-AGE indicates the freshness of the resource, it is clear that through this, observe extension clients will always have a fresh and up-to-date representation of the resource. Figure 4 shows how the observe option is used to get up-to-date resource states.

As such, when Observe is used, the CoAP client will get a notification response whenever the state of the observed resource changes or its MAX-AGE expires. For frequently changing resources or resource with a low MAX-AGE value, this results in frequent notifications, which is not ideal in constrained networks. Also, it is unclear how non-cacheable (MAX-AGE equal to 0) resources should be handled. In many cases, an observer will typically be interested in state changes that satisfy a specific condition, instead of receiving all state changes or notifications that only update the freshness.

Figure 1 CoAP message format consisting of a 4-byte base binary header followed by optional extensions.

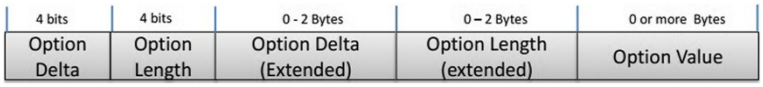

Figure 2 CoAP option format.

However, the current observe draft stresses on providing the clients with up-to-date information about the state of a resource. Applications that are interested in values that exceed some thresholds will simply drop the transmitted packets upon reception if they do not meet their criteria (client side filtering). This unnecessary data transmission can be costly to the already constrained objects. The increased number of packet transmissions in highly dynamic environment will also increase the network congestion and for larger networks the impact can be significant. In addition, the power consumption (processing, transmission, and listening) can be higher for the overall network.

Therefore, in several cases, one could benefit from a solution for subscribing to very specific events only, i.e., conditional observations. Since we are dealing with constrained devices, such a solution should satisfy several requirements. The functionality should have a sufficiently small footprint, allowing the implementation on very constrained devices. It should be usable by all resources on a constrained device without additional programming complexity. Further, it should offer sufficient expressiveness in order to be able to express conditions that are encountered frequently across resources and across IoT use cases. Finally, if needed, extensions should be possible in order to cope with future requirements. Based on these requirements, we have chosen to realize this conditional observe functionality by embedding it in the CoAP protocol as a new CoAP option. Before presenting our solution, we will first discuss related work that aims to achieve similar functionality, and position it against our approach.

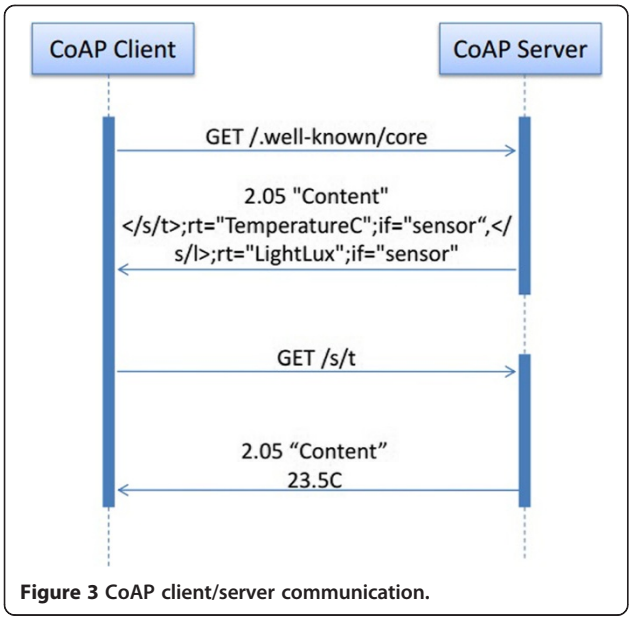

Figure 3 CoAP client/server communication.

3 Related work

There are a number of research activities under way on resource observation in WSNs. Different groups are using different approaches to come up with outstanding solutions and technologies. Publish/subscribe systems are widely used in the Internet already for a while. The basic concept of such systems is similar to normal observation where subscribers register at publishers (notifiers) and get responses depending on the original request made by the subscribers [8]. Different authors have proposed similar solutions to be used in wireless sensor networks. MQTT-S is a protocol proposed to handle publish/subscribe issues in WSNs. The protocol is based on the MQTT protocol, an established protocol for lightweight publish/subscribe reliable messaging transport, optimized to connect physical

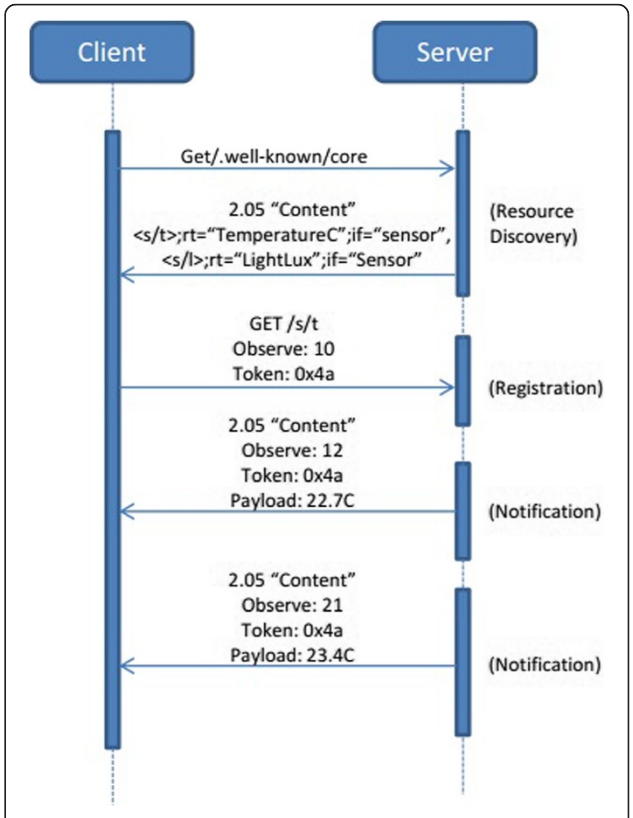

Figure 4 Normal observation.

world devices/messages and events with enterprise servers and other consumers. The protocol introduces MQTT-S gateways and forwarders to communicate publish/subscribe information between clients and the MQTT broker, which ultimately responds with the required information [9]. With this approach, a 3rd party is required to realize the desired functionality, whereas we want to allow direct end-to-end interactions with the constrained devices. There are also middleware-based pub/sub solutions such as *Mires* [10] and *PSWare* [11]. Most of these solutions introduce a new protocol specifically addressing this issue while our approach, however, is an extension of an existing protocol that is being developed in an open standardization organization. This way, our approach significantly reduces the additional memory and processing requirement for realizing this new functionality, since it builds upon functionality already present in any CoAP implementation.

The European telecommunications standards institute (ETSI) has also proposed a standard to address observation relationships in Machine-to-Machine (M2M) communications. The ETSI Machine-To-Machine (M2M) Communications functional architecture [12] states how RESTful web services can be used in M2M communications. Subscription management is one of the areas the document addresses. In the document, a client may subscribe for a specific resource or an attribute of a resource by specifying filtering criteria, if required. The ETSI standard follows its own functional architecture that is totally different from the IETF approach. Our solution is based on the work of the IETF CoRE working group.

Another related work is [13] where conditional observation requests are represented by URI queries. An important problem with this approach is its complexity. The queries that are generated may have limited readability and could be difficult to represent. Furthermore, URI queries are very resource specific complicating automatic processing of conditional observations or code reuse over several resources. Using a CoAP option for conditional observations makes this functionality independent of any specific resource implementation, whereas URI queries can be used for resource-specific functionalities. Further, the link with the Observe option is lost by spreading this functionality over both URI queries and options and the multitude of URI queries that can occur makes it more complex for intermediaries to process this information.

Finally, the Open Geospacial Consortium (OGC), Inc. has been developing different standards in the area of geospatial data. One of the standards developed by the OGC is the sensor observation service (SOS) that deals with the specifications of data observation from different sensors in different, possibly geographically scattered, sensor networks [14]. The standard specifies that a GetObservation request may have several mandatory and optional parameters. One of the optional parameters is featureOfInterest, which is similar to our observation type. However, this approach is more focused for geographical observations and is a subset of a bigger framework, which significantly differs from the IETF recommendation.

4 Conditional observe

To avoid transmission of unwanted notifications to clients, the authors of this paper have proposed a new CoAP option 'Condition' as an extension to the Observe Option in order to support conditional observations [15]. This option can be used by a CoAP client to specify the conditions the client is interested in. Now, only when the condition is met, the CoAP server will send a notification response with the latest state change. When the condition is not met, the CoAP server will not send the notification response. Figure 5 shows the operation of conditional observation.

The Condition option has to be used in combination with the Observe option and can be used both in request and response messages. In a GET request message, the Condition option represents the condition the client wants to apply to the observation relationship. It is used to describe the resource states the client is interested in. In response to the initial GET request message, the Condition option, together with the Observe option, indicates

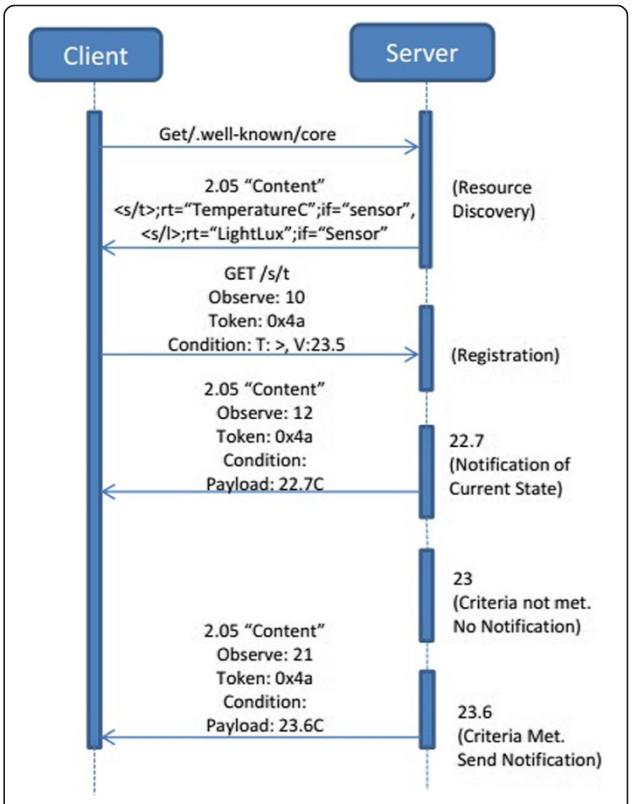

Figure 5 Conditional observation.

that the client has been added to the list of observers and that notifications will be sent only when the resource state meets the condition specified in the Condition option. In all further notifications, the Condition option identifies the condition to which the notification applies. In the following subsections we will further describe the semantics and usage of the Condition option, illustrating the capabilities of this extension.

4.1 The Condition option format

The Condition option is an elective and proxy unsafe option [4,15]. The Option (see Figure 6) may have length between 1 and 5 bytes. The most significant 5 bits of the first byte indicate the condition type allowing up to 32 different condition types; the following bit is reliability flag indicating if the response should be acknowledged or not and the last two bits indicate type of the value in the following bytes. Currently, integer, float, and duration are identified as condition value types. The subsequent bytes, which are optional, store the conditional values to be exchanged.

4.2 Condition types

Li et al. [15] identified nine condition types, some of which are time-based while others are value-based. Minimum Response Time, Maximum Response Time, and Periodic option types are time-based conditions, whereas AllValues<, AllValues>, Value=, Value<>, and Step use the sensor reading values as notification criteria. The Time Series condition type is neither related to time nor to sensor readings.

To further illustrate how different condition types generate notifications, we show an example where a client and a server node establish a temperature observation relationship. Sensor readings drawn every 5 s will be notified to the client depending on various conditions. Figure 7 shows the temperature (in °C) and the time the data is drawn from the sensors. In the figure, the triangles indicate the sensor reading values. For instance, the graph shows that when the first GET request was sent (at Time 0), the temperature was 22, and after 5 s, the value is still the same. The next figure (Figure 8) represents which notifications are generated for different condition types using small diamonds. For the purpose of this illustration, the CoAP MAX-AGE option value is set to the default value of 60 s. This means that, for normal observe, the client must be notified if the last notification was 60 or more s ago irrespective of the resource state change, as described in [5]. For the sake of comparison, we will first present the notification trend when using normal observe.

4.2.1 Normal observe

According to the CoAP draft document [4], a server sends notifications to observers in three cases: first, a notification is sent to clients when the observation relationship is established for the first time to indicate that the client is added to the observers list; second, whenever the resource state changes the server sends notifications;

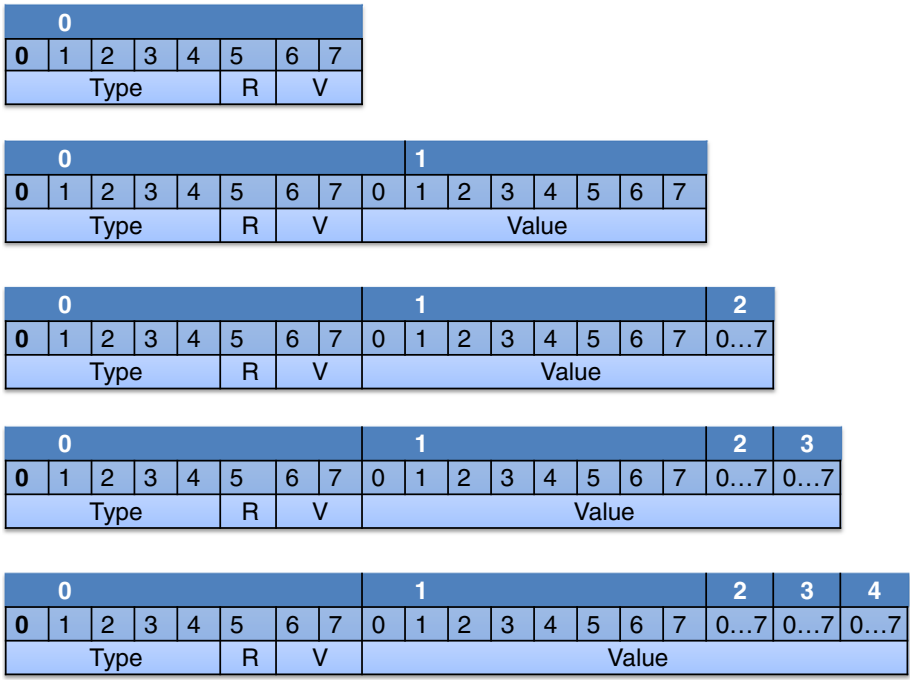

Figure 6 Format of the option value of the Condition option.

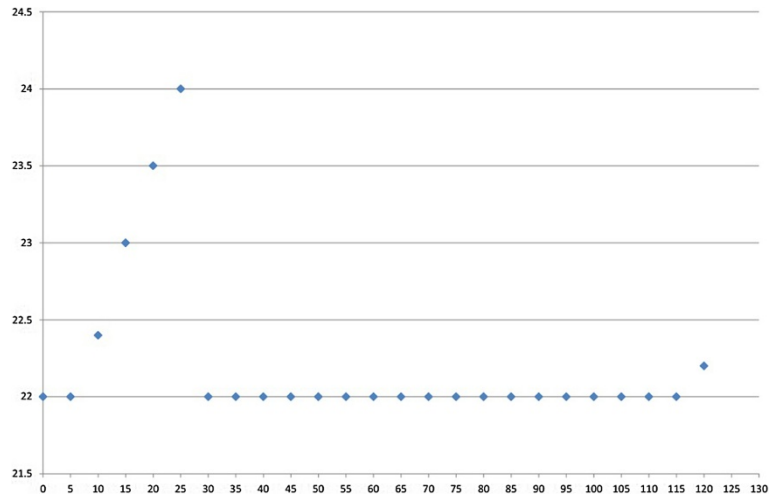

Figure 7 Temperature (°C) data over 120 s.

finally, a notification is also sent when the data previously sent to the client is not fresh as indicated by the CoAP MAX-AGE option which by default is set to 60. In such cases, the server sends the notification, if the previous notification is older than the MAX-AGE value (even if the resource state stays the same).

Accordingly, given the values in Figure 7, the server sends notifications at the establishment of the observation relationship (at time 0) every time the value changes (at times 10, 15, 20, 25, 30, and 120 s), and the MAX-AGE expires (at time 90 s) as shown in the top row of Figure 8.

4.2.2 Condition type 1: time series

With Time Series condition type, every change of resource state triggers notification. The notification criteria are similar to normal observation. The only difference is that Time Series option ignores the CoAP MAX-AGE option while normal Observe sends a notification when the MAX-AGE timer expires.

The T-Series row of Figure 8 shows the packet transmission for conditional observation type Time Series. According to the sensor readings of Figure 7, the client is notified at time 0 (during establishment of the

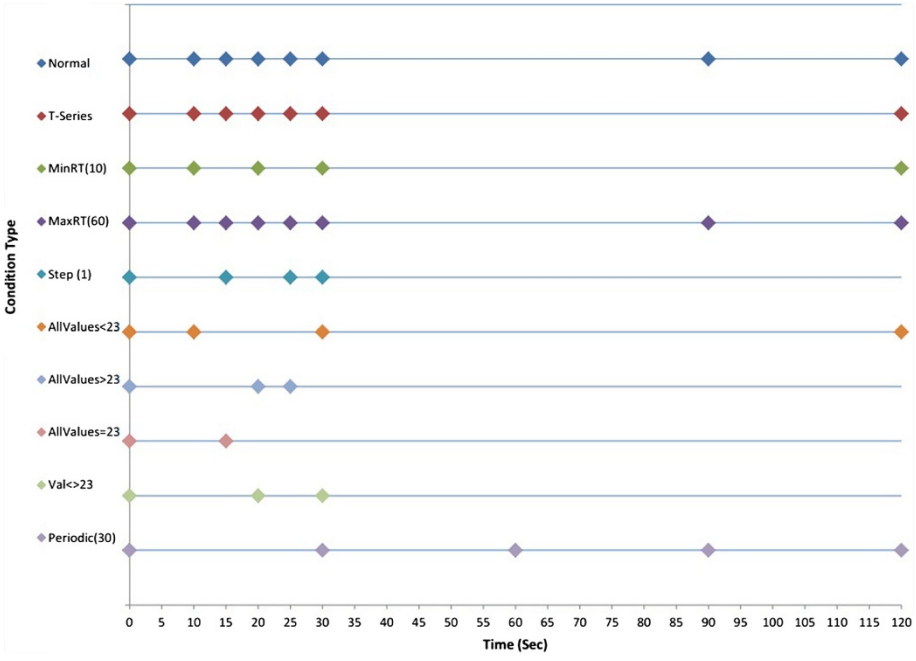

Figure 8 Notifications generated while using different condition types.

relationship) and at times 10, 15, 20, 25, 30, 120 s (when the resource state changes).

4.2.3 Condition type 2: minimum response time (MinRT)

When the condition type Minimum Response Time (MinRT) is used in observation relationships, the server sends notification by leaving a fixed minimum amount of time between successive notifications. This condition type is highly valuable for systems where the value changes up and down very frequently and the observer is not interested in every change. Consequently, the server does not always send notifications every time the resource state changes.

The MinRT (10) row of Figure 8 shows a relationship where the client requests the server to be notified about state changes, but leaving at least 10 s between notifications. In this case, the client sends notifications at time 0 (during establishment of relationship), at times 10, 20, 30, and 120 s. If we closely look at the values at times 15 and 25 s, the values are changed after previous notifications, but since the difference between the current time and the last notification time is less than 10 seconds, there will be no notification sent to the clients at those times. Also note that, the MAX-AGE option has no impact here.

4.2.4 Condition type 3: maximum response time (MaxRT)

For this condition, the value specified in the condition value field gives the maximum time in seconds the server is allowed to leave between subsequent notifications. What this means is that the server has to send notifications in three cases: first, just like all other condition types and normal observation, at the beginning of the observation relationship; second, whenever there is a resource state change; and third, when there is no state change but the maximum response time is reached.

The MaxRT row of Figure 8 shows the notification pattern for a client requesting notification by setting Maximum Response Time to 60 s. Accordingly, the server notifies the client at time 0 (initial notification), at times 10, 15, 20, 25, 30, and 120 s (notification due to value changes), and at time 90 s (notification due to maximum response time). This condition type, in a way, is similar to normal observe with MAX-AGE set to 60.

4.2.5 Condition type 4: step

Depending on the environment where the server node is deployed, the state of a resource might change so frequently that excessive packets are generated. However, the changes may not be significant enough for the client to trigger any action. In such cases, the client may inform the server to send notifications only when the change is more than a specific value by using the Step condition type.

In the Step (1) row of Figure 8, the client informs the server to send notifications only when the change in value is greater than or equal to 1. As a result, notifications are only sent at time 0, 15, 25, and 30 s. Since the other changes are not significant enough, the server does not send notifications.

4.2.6 Condition type 5: all values<

In many cases, clients are not interested in state changes which result in values above a specific threshold. For example, to turn on a heater, the temperature should be below a specific threshold. In such cases, the sensor node responsible to regulate the behavior of the heater is not interested in values which are above the threshold. Hence, they may indicate this preference by using the AllValues< (All values less).

In the AllValues< row of Figure 8, we can see that the client is interested to get notified only when the resource state changes result in value below 23. Thus, notifications are sent at times 0, 10, 30, and 120 s.

4.2.7 Condition type 6: all values>

This condition type is similar to condition 5 above. The only difference is that the notification is sent only when the new value exceeds a threshold set by the client. As Figure 8 illustrates, the client is interested to receive notifications only when the resource state is changed and the resulting value is above 23. Consequently, the server sends notifications at times 0, 20, and 25 s only.

4.2.8 Condition type 7: value=

This condition indicates that a client is only interested in receiving notifications whenever the state of the resource changes and the new value is equal to the value specified in the condition value field. In our example of Figure 8, the client is interested in values equal to 23. This means that the server has to send notifications only when the value changes, and the new value is 23. Therefore, the notifications are sent at times 0 and 15 s only.

4.2.9 Condition type 8: value<>

Some applications might require the values they are monitoring to be constant. In health care system, machines at intensive care units (ICUs) the machines that monitor a patient's vital signs could be a good example. In such cases, there are vital signs including body temperature, heartbeat, and blood pressure that must be constant, showing that the patient is in a good condition. However, if the values differ from the specified value, it might indicate the patient needs attention. The Value<> (value different from) condition type indicates that the client should be notified when the value changes and is below or above the specified threshold. Once the notification has been sent, no new notifications are sent for subsequent state changes

where the value remains higher or lower. As such, a single notification is sent whenever a threshold is passed in either direction.

The Value<>(23) row in Figure 8 shows that the client needs to be notified only when resource state changes result in values other than 23. As a result, notifications are sent at times 0, 20, and 30 s: notification at time 0 is the initial transmission; notification at time 20 s is sent because that was the first change that deviates from 23 (and it was below 23); and notification at time 30 s was sent because it was the first time the value goes below 23. The value changes at 10 s was not sent because it is still below the threshold, and value at 25 was not notified because it is still above the threshold (which was notified at time 20).

4.2.10 Condition type 9: periodic

Many environment monitoring applications may require receiving notifications periodically despite the resource state change. Such applications may use the Periodic condition type along with the period of notification. The Periodic(30) row of Figure 8 shows notification trends where a client requires to be notified every 30 s. In this example, notifications are sent at times 0, 30, 60, 90, and 120 s.

One can see clearly that, depending on the condition of interest, a different number of notifications will be transmitted over the constrained network. The exact number will depend on the condition type and, if present, the value in the condition option.

5 Implementation

Our implementation of conditional observation is based on *Erbium (Er)* - a low-power REST engine for Contiki developed by Matthias Kovatsch together with the *Swedish Institute of Computer Science*. The Erbium REST engine includes a CoAP implementation that supports CoAP drafts 03, 07, 12, and 13. It also supports block-wise transfers and resource observation [16]. To support normal observation, Erbium employs two different mechanisms at the server side. The first mechanism uses timers that are used to periodically check states of resources and notify observers whenever there is a state change. The other mechanism is event-based. Whenever an event (e.g., change in temperature) occurs, an event handler will be called, which, in turn, calls a function that notifies registered observers. In the remainder of this paper, we have used timer based, periodic checks for resource changes.

We extended this CoAP implementation to support the new Condition option and provided some resources that allow conditional observations. Figure 9 is a high-level architectural diagram of Erbium running on a server node handling normal or conditional observation. The architecture consists of several components, namely, the resources, the REST engine, the CoAP (and/or HTTP framework), and optional modules such as (Conditional) Observe Module. The REST engine is responsible for initializing the CoAP framework to store a list of activated resources and to communicate with the optional modules. A conditional observe request received by the CoAP framework will be handled by a service callback function which is declared in the REST engine. The REST engine uses the corresponding handler function to access the states of the resources. As the request is an observation request, the client needs to be registered as an observer in the Conditional Observation Module for future notifications by calling a Post Handler function. The generation of the first response and subsequent notifications are handled by the CoAP framework. For subsequent notifications, the registration of a single observer will trigger the activation of a function that periodically checks for resource state changes and informs all registered observers. The period is defined for each observable resource separately upon initialization of the resource.

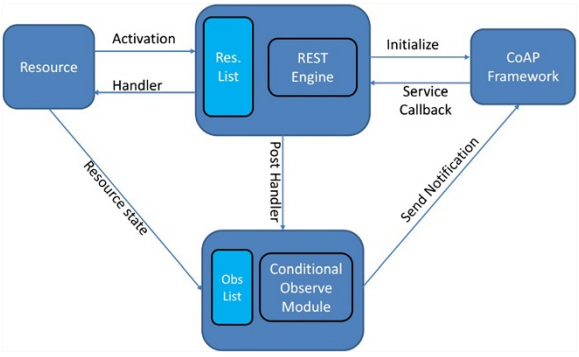

Figure 9 Architecture of Erbium.

Figure 10 is a zoomed-in architectural diagram of the Conditional Observation Module. When a client sends a Conditional Observation request, the CoAP framework will receive it. The framework, after confirming that it is a GET request, will call a callback function in the REST engine. Upon receipt of the request, the REST engine does two things: first, it prepares the first response by using the predefined handler function and calls a post handler function to add the observer to the observer list. For each observer, the IP address, port number, URI, and refresh timer are stored for normal observe, while for conditional observation also condition information, last notified value and last notification time are stored. The REST engine then periodically checks for resource states and calls the Notify Observer function (which is part of the Conditional Observe Module) to check if the new value satisfies the filtering criteria set by the client. If it does, the CoAP framework sends the notification to the respective observer(s). Similarly, if a client wishes to stop an observation relationship, it sends a normal GET request to the specific resource which will be received by

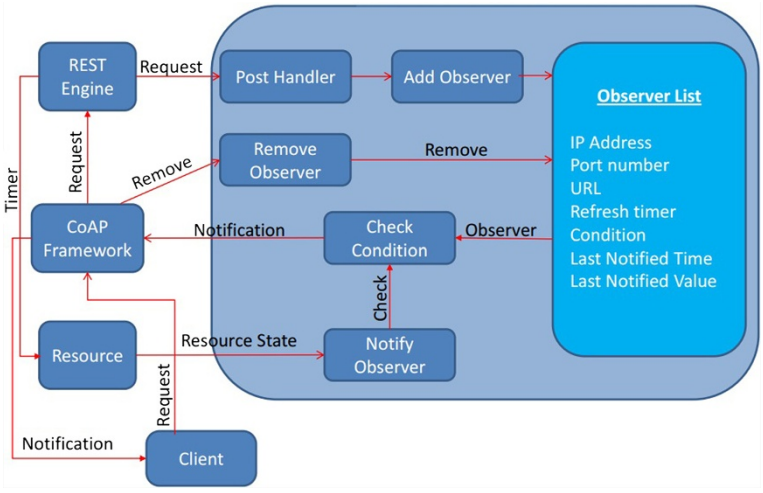

Figure 10 Conditional Observation Module.

the CoAP framework and will be sent to the Conditional observe module to be removed from the list.

One of the constraints of smart objects is memory. One may wonder about the changes we needed to make to the existing implementation and the code overhead introduced to achieve this additional functionality. As mentioned above, the original Erbium Implementation supports Normal Observe [5]. Our implementation requires additional RAM to store additional observers' information such as observation condition (condition type, value type, reliability flag, and condition value), last notification time, and last notified value. In addition, it requires more ROM to store instructions that are used to check if the new resource state satisfies the specified condition. Table 1 shows the TEXT, Data, and BSS section requirements of both Normal Observe (the original implementation) and Conditional Observe. Note that the conditional observe implementation also encompasses normal observe functionality.

It can be seen from the table that the Text segment (ROM) requirement for Conditional Observe is slightly larger than Normal Observe. Similarly, the size of the BSS segment, which stores uninitialized variables, is larger in case of conditional observe just by a few bytes. As our findings in the following sections illustrate, 720 bytes overhead is affordable for the advantage that can be gained through the use of conditional observation, either from a performance viewpoint or from an application developer viewpoint.

Table 1 Memory requirements of Normal Observe and Conditional Observe

	Text (Byte)	Data (Byte)	BSS (Byte)	Total (Byte)
Normal Observe	50,398	386	6,050	56,834
Conditional Observe	51,096	386	6,072	57,554
Delta	698	0	22	720

6 Evaluation

6.1 Scenario 1: basic evaluation

We used different scenarios to illustrate the relevance of Conditional Observe as an extension to the Normal Observe functionality of CoAP. In the first set of experiments, we used Zolertia Z1 motes to be used as client and server nodes in Cooja. To capture the impact of network size on performance, we used between 0 and 6 intermediate Z1 nodes, which merely exist to act as routers between the client and server nodes. We selected the AllValues> (value-based) and Periodic (Time-based) condition types to compare the performance against Normal Observe. For every hop, and every condition value, we run the test 10 times to average the results. For sensor values, we generated 288 pseudo-random numbers between 17 and 26 (representing temperature values). The average of the values is 20. For AllValues> condition type, we tested three condition values: the minimum (17), the average (20), and the maximum (26). Every 5 s, the server is made to retrieve a value from an array of 288 numbers sequentially as a new sensor reading. As CoAP supports both confirmable and non-confirmable requests, we repeated the same experiment twice to see the impact of reliable communication on network performance.

Figure 11, shows the number of packets transmitted for different condition types, while Figure 12 shows the power consumption where the requests are sent as non-confirmable. Figure 13 shows the power consumption in the case of confirmable communication.

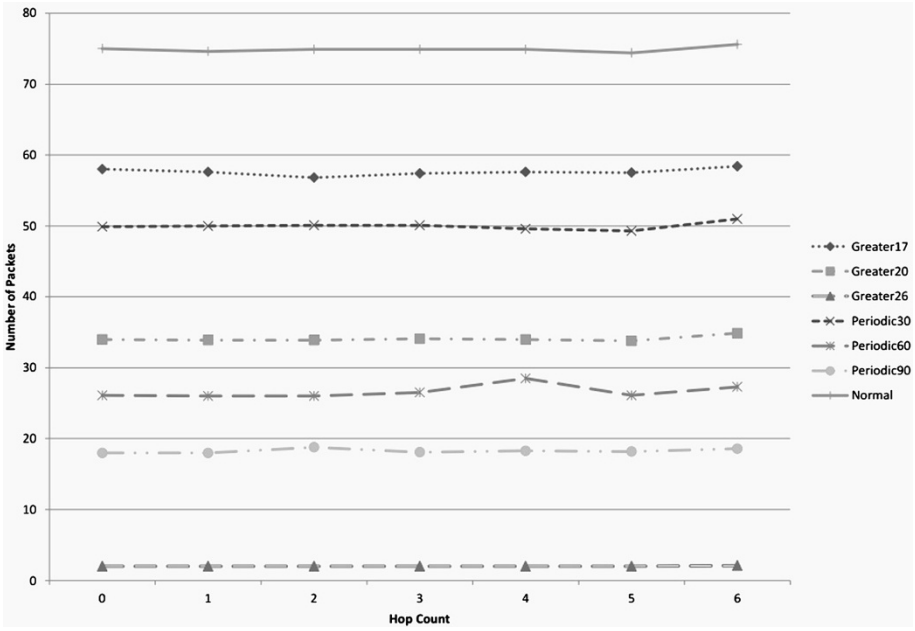

Figure 11 Number of packets transmitted vs. hop count.

We may learn two basic lessons from Figures 11, 12, and 13: first, this simple scenario shows that the solution we proposed works and is implementable in constrained devices such as Z1 motes; second, for such a simple scenario, using Normal Observe as mechanism to collect all resource state change in combination with client side filtering generates a larger amount of packets as compared to all other conditional observation methods. This leads to higher power consumption and, hence, low battery life. From these findings we can conclude that, even though the exact impact is heavily dependent on specific use cases, conditional observation can be considered a useful extension to normal observation.

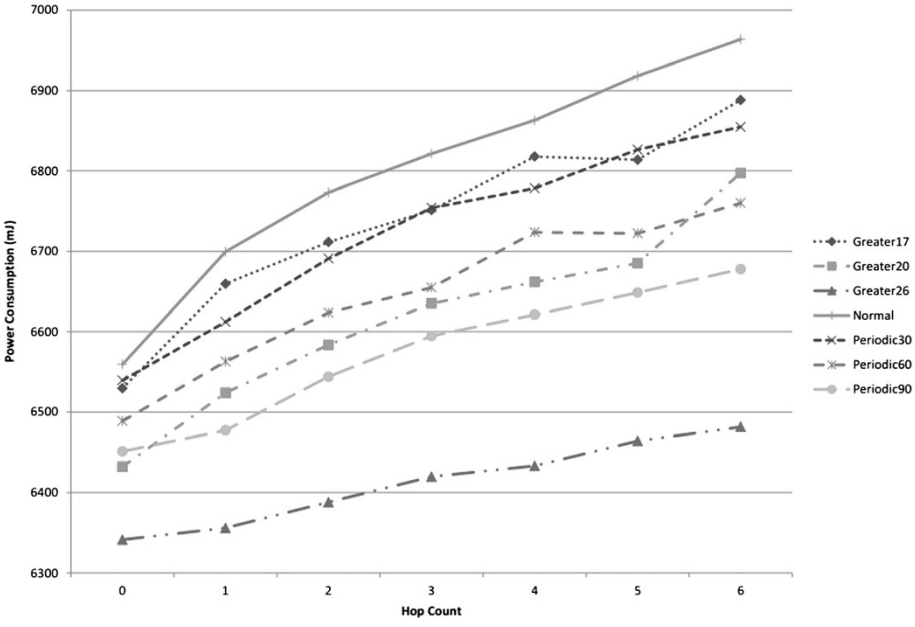

Figure 12 Power consumption vs. hop count (non-confirmable communication).

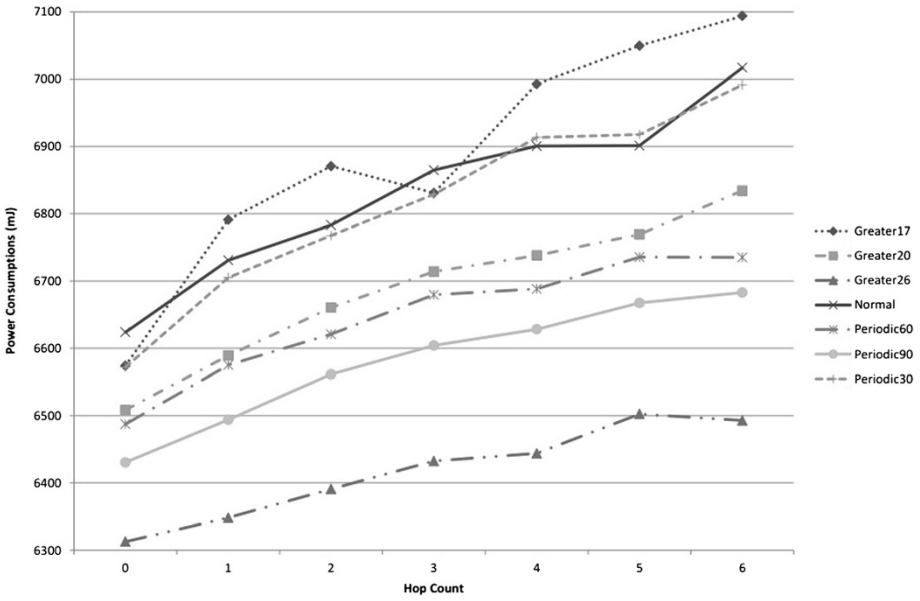

Figure 13 Power consumption of nodes (confirmable transmission).

6.2 Scenario 2: non constrained client - gateway – multiple servers

In most applications, the clients run on non-constrained devices such as normal computers, smart phones, or similar other devices. To illustrate the use of conditional observation in scenarios where multiple constrained servers are communicated from a non-constrained device and network, we combined Cooja servers running Erbium with our COAP++ client, part of our modular C++ CoAP framework developed in Click Router [17]. We used two servers and a border router running Contiki in Cooja. We established a tunnel between the boarder router and the computer so that the COAP++ client can communicate with the Cooja servers. Figure 14 shows the connection between the sensor nodes and click. As in the case of the above test, Z1 motes were used in Cooja and AllValues> condition type was used to compare the result with Normal Observe. The data to be generated by the servers is the same pseudo-random numbers used in the previous tests. The test was run 10 times with condition value 20, which is the average of the data set. The test showed that the average power consumption of the two servers is *4787 mJ* and *4936 mJ* for normal observation and *4672 mJ* and *4856 mJ* for conditional observation. These results confirm the results we achieved in the previous scenario.

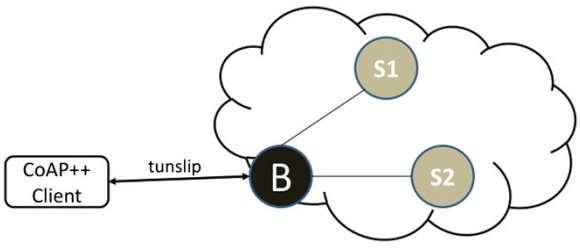

Figure 14 Experimental setup consisting of non-constrained client and two constrained servers.

The tests above show that the new implementation can be run in constrained devices such as Zolertia Z1 motes which have 8-KB RAM and 92-KB ROM. The slight increase in memory requirement, especially to filter packets on the server, is an affordable trade-off to gain substantial energy saving and avoid congestion of the constrained networks.

6.3 Mathematical evaluation

The previous tests show that conditional observation can achieve a significant gain in terms of the reduction of the number of transmitted packet and subsequently of the reduction in power consumption of constrained devices. Of course, the exact gain depends largely, amongst others, of the frequency of resource states and the specific conditions interested in. In this subsection, we will show this potential gain of conditional observation through a mathematical evaluation.

The total power consumed in a given period by a device, E, is the sum of the consumption of the radio and the microcontroller chips. For simplicity, we will assume that the power consumption of other peripheral devices, such as sensors, is insignificant. Therefore,

$$E = E_{\text{Radio}} + E_{\text{Processing}}.$$

The Radio could be either in active transmitting (Tx), active receiving (Rx), and idle transmitting or idle listening

state, and the microcontroller will be either in Active Mode (CPU-Active) or Low Power Mode (CPU-LPM). Assuming that the radio consumes equal amount of power during idle listening and idle transmitting state, we have:

$$E = E_{\mathrm{Tx}} + E_{\mathrm{Rx}} + E_{\mathrm{IDLE}} + E_{\mathrm{CPU-Active}} + E_{\mathrm{CPU-LPM}}.$$

Energy, power consumption in a given period of time, can be computed as,

$$\text{Energy} = \text{Power} \times \text{Time}.$$

Therefore, the total power consumption in a particular period of time will be:

$$E = P_{\mathrm{Tx}} \times T_{\mathrm{Tx}} + P_{\mathrm{Rx}} \times T_{\mathrm{Rx}} + P_{\mathrm{IDLE}} \times T_{\mathrm{IDLE}} + P_{\mathrm{CPU-Active}} \times T_{\mathrm{CPU-Active}} + P_{\mathrm{CPU-LPM}} \times T_{\mathrm{CPU-LPM}}$$

and

$$T_{\mathrm{Tx}} + T_{\mathrm{Rx}} + T_{\mathrm{Idle}} = T_{\mathrm{CPU-Active}} + T_{\mathrm{CPU-LPM}}.$$

The power consumption values are all known from the datasheets of the chips of the constrained devices, the time values for the radio can be derived from the MAC protocol and the number of packet transmissions and receptions and the time values for the CPU can be derived from experimental results. To compare the power consumption difference between Normal and Conditional Observe and to keep the mathematical model sufficiently simple, we make the following assumptions:

- We use value-based condition types.
- The resource state changes every S seconds.
- The device uses a duty-cycled low power listening protocol. The length of the duty cycle is L, and the duty cycling value is d (only d% of the time the radio is active, listening for incoming packets in order to save energy). When the device needs to transmit a packet, it has to turn on the radio for the entire period L.
- The probability that the condition is not fulfilled is p.
- The value of MAX-AGE is equal to 60 s.
- The transmission of a notification requires only a single packet.
- The notifications are non-confirmable messages, so the server generating the notifications is not receiving any packets.

For Normal Observe, every S seconds the device will do some processing and transmit a notification. In case S becomes larger than Max-Age, a notification is also sent every time MAX-AGE expires. This brings the total number of notifications sent, N, equal to CEIL (S/Max-Age).

Hence, for Normal Observe, $T_{\mathrm{CPU-LPM}}$ is:

$$T_{\mathrm{CPU-LPM}} = S - T_{\mathrm{CPU-Active}}.$$

For Conditional Observation, the transmission of a packet in the interval S depends on the condition value and is probabilistic. We know that conditional observations require some additional processing for checking the condition. However, in case no notification must be sent, no processing is needed to prepare the packet transmission. To incorporate both effects, we introduce Δ_{Proc} giving an estimate for the difference in processing. Thus, the time, $T'_{\mathrm{CPU-LPM}}$ is:

$$T'_{\mathrm{CPU-LPM}} = T_{\mathrm{CPU-LPM}} - \Delta_{\mathrm{Proc}} = S - T'_{\mathrm{CPU-Active}} = S - (T_{\mathrm{CPU-Active}} + \Delta_{\mathrm{Proc}}).$$

Similarly, the time the radio chip spent in the different states, can be computed from the period S, the duty cycle length L the duty cycling value d and the number of packets to be transmitted. For Normal Observation, the time for TX, RX, and Idle states is given by:

$$T_{\mathrm{Rx}} = (S - N \times L) \times d$$

$$T_{\mathrm{IDLE}} = (S - N \times L) \times (1 - d)$$

$$T_{\mathrm{Tx}} = N \times L = S - (T_{\mathrm{IDLE}} + T_{\mathrm{RX}})$$

Here, we have assumed that if a device has to transmit packets, it will use the whole period L for the transmission. The formula for conditional observation will have to take the probability of transmission into consideration. For probability value p (condition not fulfilled), the time spent will be:

$$T_{\mathrm{Tx}} = 0$$

$$T_{\mathrm{IDLE}} = S \times (1 - d)$$

$$T_{\mathrm{Rx}} = S \times d$$

For probability values 1 to p (condition fulfilled), the overhead will be the same as normal observation, with N equal to 1. Therefore, the total energy consumption, based on our equations above, during S seconds for Normal Observe, O(S) and Conditional Observe, CO(S) will be:

$$
\begin{aligned}
E &= P_{\mathrm{Tx}} \times T_{\mathrm{Tx}} + P_{\mathrm{Rx}} \times T_{\mathrm{Rx}} + P_{\mathrm{IDLE}} \times T_{\mathrm{IDLE}} \\
&\quad + P_{\mathrm{CPU-Active}} \times T_{\mathrm{CPU-Active}} + P_{\mathrm{CPU-LPM}} \\
&\quad \times T_{\mathrm{CPU-LPM}} O(S) = P_{\mathrm{CPU-Active}} \times T_{\mathrm{CPU-Active}} \\
&\quad + P_{\mathrm{CPU-LPM}} \times (S - T_{\mathrm{CPU-Active}}) \\
&\quad + P_{\mathrm{Tx}} \times N \times L + P_{\mathrm{Rx}} \times (S - N \times L) \times d \\
&\quad + P_{\mathrm{IDLE}} \times (S - N \times L) \times (1-d) CO(S) \\
&= P_{\mathrm{CPU-Active}} \times (T_{\mathrm{CPU-Active}} + \Delta_{\mathrm{Proc}}) + P_{\mathrm{CPU-LPM}} \\
&\quad \times (S - T_{\mathrm{CPU-Active}} - \Delta_{\mathrm{Proc}}) + p \\
&\quad \times [P_{\mathrm{Tx}} \times 0 + P_{\mathrm{Rx}} \times S \times d + P_{\mathrm{IDLE}} \times S \times (1-d)] \\
&\quad + (1-p) \times [P_{\mathrm{Tx}} \times L + P_{\mathrm{Rx}} \times (S-L) \times d + P_{\mathrm{IDLE}} \\
&\quad \times (S-L) \times (1-d)]
\end{aligned}
$$

In order to be able to evaluate the above values for different parameter values of S and p, we used the parameters shown in Table 2.

From the values O(S) and CO(S), one can easily calculate the energy consumed during 1 s and compare the difference in energy consumption between Normal Observe (assuming the collection of all values using Normal Observe and client side filtering) and Conditional Observe. Figure 15 shows the reduction in energy consumption that can be achieved this way, for different values of S and p.

We see that for increasing values of the probability p, i.e., the probability that the condition is not fulfilled, the reduction in energy consumption also increases. For a fixed time S between resource changes, and thus a fixed amount of potential notifications, an increasing probability p implies that less notifications have to be sent compared to normal observe, leading to a reduction of the energy spent on transmitting these notifications and thus of the overall energy consumption. This is in line with our experimental evaluation. Further, we notice that, for a fixed value of p, smaller values of S, the time between resource changes, leads to major reductions in energy consumption. On the other hand, for larger values of S, the potential energy reduction gradually decreases as can be seen from the zoomed-in area of the chart. For values larger than 30 s the average reduction varies between approximately 2% and 6%. The fact that the curves periodically rise and fall is due to the impact of the MAX-AGE value, which causes the number of notifications to be sent by Normal Observe to be a step function expressed by CEIL(S/MAX-AGE).

Table 2 Parameter values

Parameter	Value	Explanation
$I_{\mathrm{CPU-LPM}}$	0.0005 mA	Taken from the data sheets of the Z1 mote.
$I_{\mathrm{CPU-Active}}$	8 mA	
I_{Rx}	18.8 mA	
I_{Tx}	17.4 mA	
I_{IDLE}	0.426 mA	
V	3 V	
$P_{\mathrm{CPU-LPM}}$	0.0015 mW	Power = Current × Voltage
$P_{\mathrm{CPU-Active}}$	24 mW	
P_{Rx}	56.4 mW	
P_{Tx}	52.2 mW	
P_{IDLE}	1.278 mW	
L	125 ms	Based on ContikiMAC LPL
d	0.01	
Δ_{Proc}	1 ms	Approximation based on extensive experiments.
CPU active	1.48%	Average percentage of time the CPU is active, based on extensive experiments.

The impact of S on the energy reduction can be further explained by looking at the contribution of all different energy consumers to the overall energy consumption as shown in Figure 16. For frequently changing resource values and thus more frequent notifications for a given value of p, the power consumed for transmitting notifications makes up a large part of the total energy budget. For increasing values of S, and thus less notifications, this part becomes smaller and smaller compared to other energy consumers such as the idle energy consumption of the radio. Since conditional observe almost solely impacts the TX part of the energy budget, its impact is reduced for larger values of S. Of course, it should be noted that the frequency with which values (e.g., sensor readings) can change also depends on the granularity of the measurements and/or the application on the device. For instance, the device can be programmed to retrieve the latest sensor reading only every 5 min, but it could also read out the temperature every 10 s. Further, if the granularity of the readings is higher, e.g., a granularity of 0.1 Centigrade instead of 1 Centigrade, readings will more often result in notifications.

The above mathematical evaluation reveals that a mechanism such as Conditional Observe is extremely useful for resources that change very frequently. One could (falsely) conclude that it is not that useful for larger values of S. However, this is not true. First of all, the concept of conditional observations remains very useful for application developers, which are now offered easy to use primitives to collect sensor data based on conditions, and can serve as an enabler for IoT applications. Next to this, the above mathematical evaluation has been made in the assumption of a single server with a single resource that sends its notifications directly to the sink or gateway of the sensor network. In case a single constrained device hosts multiple resources (temperature, light, humidity,…) that are conditionally observed, one will experience the combined effect of having less packet transmissions for every resource individually. This means

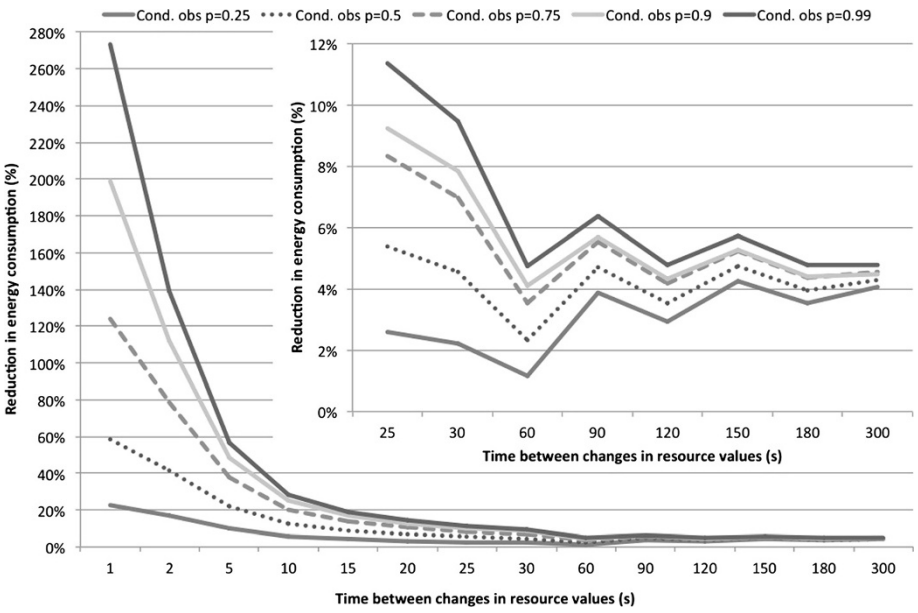

Figure 15 Power consumption vs. probability of packet transmission.

that even for larger values of S a significant energy reduction can be achieved. This effect is illustrated in Figure 17. When the number of resources per device is increased, all resources being observed conditionally and having their state changed every S seconds, the reduction in energy consumption remains significant even for higher values of S. It is also worth noticing that the gain for S equal to 60 is smaller than the gain for larger values of S. This is because of the impact of MAX-AGE on the number of notifications sent using normal observe.

Further, in case one of the resources has a smaller MAX-AGE value than the default value of 60 s, the reduction also becomes bigger: between 2% and 6% for S values between 30 and 300 s and MAX-AGE equal to 60 s; between 4% and 10% for S values between 30 and 300 s and MAX-AGE equal to 30 s; and between 20% and 30%

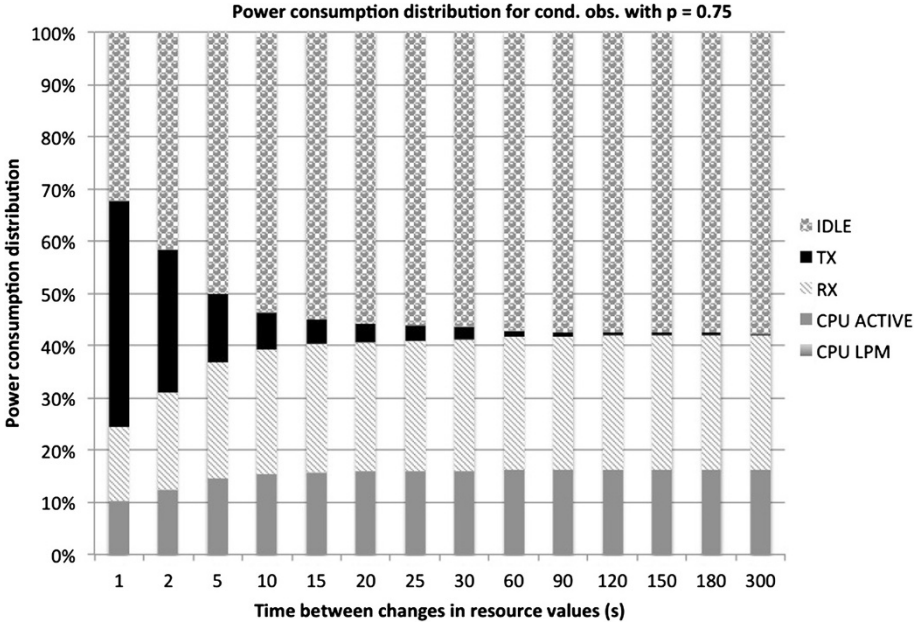

Figure 16 Distribution of the power consumption over all different energy consumers.

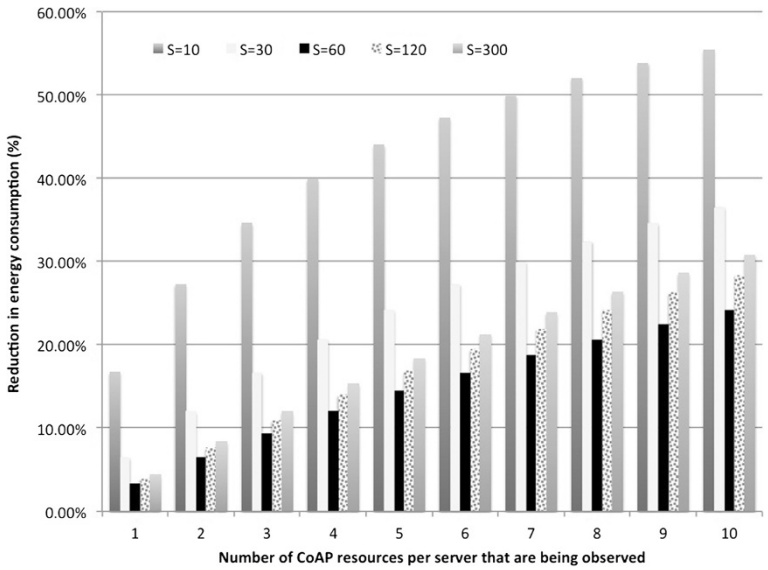

Figure 17 Energy consumption reduction of conditional observation ($p = 0.75$) vs. normal observation for varying number of resources.

for S values between 30 and 300 s and MAX-AGE equal to 10 s. Last but not least, there is also the effect of larger-scale, multi-hop networks with multiple servers. In this case, every single notification will result in multiple packet transmission through the network, increasing the TX energy consumption in all intermediate nodes and thus contributing to the overall reduction of the network lifetime.

A last aspect that has not been discussed so far is the impact of multiple conditional observation relationships on a single resource (i.e., by different clients). For normal observe and in the presence of an intermediary, the intermediary can aggregate multiple observe relationship into a single one. This means that the intermediary establishes an observe relationship itself on behalf of multiple clients and delivers the resulting notifications to all clients. This way, notifications have to travel through the network only once. When using conditional observations in the presence of an intermediary, the possibility to aggregate different conditional observations into a single conditional observation relationship strongly depends on the condition type and associated values. In this case, it is possible that no optimal aggregation can be found, reducing somewhat the overall performance.

7 Use cases

From the previous discussion, it has become clear that the actual advantage in terms of performance gain (energy reduction) of conditional observations is heavily dependent on specific-use cases. Apart from that, there is the additional advantage that it offers easy-to-use primitives to collect sensor data of interest, which have a small footprint and which are reusable by all CoAP resources hosted on a device. To concretize the advantage of having conditional observation as an extension to CoAP, we will now discuss in more detail three real-life IoT-use cases.

7.1 Heating and cooling systems

Smart buildings, heavy machineries, and greenhouses use temperature sensors to regulate their environment. Input from sensors will be used by heating and cooling systems, i.e., the actuators, to take the necessary actions to maintain the temperature at a specified level. If the temperature exceeds beyond a certain level, cooling systems at particular locations need to be activated to lower the temperature. Similarly, if it is below a certain threshold, a heating system has to increase the temperature.

When using wireless embedded systems to monitor and control the environment, conditional observation may play a significant role in such systems. Consider a building equipped with such a system. If the temperature in the building has to be maintained between 19°C and 22°C, the sensors have to inform either the heater or the cooler if the temperature is out of this range. To realize this, the heating or cooling system that is linked to a sensor may send two conditional observation requests. The first condition type should be AllValues < 19 and the second should be AllValues > 22. When the server receives these requests, on separate port numbers, it stores them as two different requests and sends notifications to the client when one of the two conditions is fulfilled.

To test this scenario using our implementation, we performed a test involving two instances of CoAP++

clients, a Cooja Border Router, Cooja CoAP server, and two intermediate nodes, as shown in Figure 18. We used real-life temperature data collected at Intel Laboratories. To simplify the test, we used a 1-day data collected every 5 min starting from midnight. The 288 data points were sent by the CoAP server every 5 s when the simulation starts. We repeated the experiment 10 times to average the result. The final result shows that, Normal Observe consumed *4,936 mJ* while Conditional Observe consumed *4,716 mJ* showing an improvement in power consumption by using conditional observation. The above example can be easily extended to other smart building applications involving a variety of sensors that need to be observed.

7.2 Smart environment monitoring

There is a growing concern of pollution everywhere in the world. Pollution, be it air pollution, water pollution, or land pollution, is the introduction of harmful substances to clean sources (air, water, etc.). Air quality index (AQI) is widely used to measure the level of pollution of air by different pollutants such as ground-level ozone, particulates, sulfur dioxide, carbon dioxide, carbon monoxide, and nitrogen monoxide. The AQI values fluctuate substantially depending on various situations. Most countries divide the AQI values in different categories and take different actions depending on the level of AQI.

Smart environment monitoring applications make use of WSNs to proficiently monitor the pollution level and come up with the AQI level of the environment. Such applications can also benefit from conditional observations in order to realize the desired behavior. Consider a simple environment monitoring system aimed at collecting the concentration of pollutants (e.g., CO_2) in a particular area and communicating it to a central station. This solution may be implemented in various ways. One such implementation is depicted in Figure 19. Sensor nodes (servers) are connected with each other and to the gateway node through wireless links. The gateway connects the central station with the servers. Every sensor node collects the data and sends it to the gateway node, which, in turn, communicates it to the central station. If no (conditional) observation is employed, the gateway would have to poll values from the sensors every fixed interval or whenever the need arises.

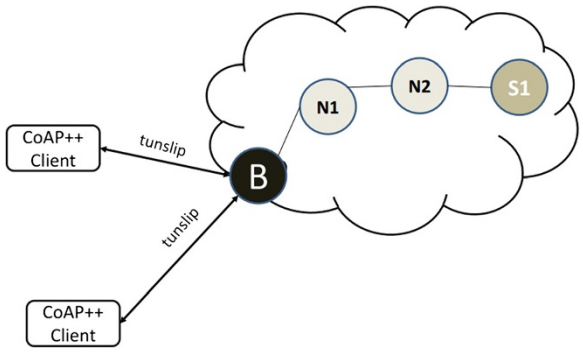

Figure 18 Experiment setup with two Click++ clients, a Contiki border router, a Contiki server and two intermediate nodes.

Using conditional observation, much more flexibility is introduced and system efficiency can be. The client (e.g., the gateway or an application in the cloud) may establish conditional observation with the servers (sensor nodes) stating its interest to be notified periodically. Here, the *periodic* condition type can be used for the subscription. Once this observation relationship is established, the servers will generate notifications periodically. It is also possible to further refine notifications based on prevailing circumstances. For example, in normal situations, where the concentration of pollutants is very low, there is no need to send notifications to the gateway very frequently. So, the notification interval can be set, for instance, to 1 h. However, as soon as the pollution level increases, which can be detected by establishing another conditional observation, the client may opt for more frequent updates (say every 5 min) by sending an updated conditional observation request which eventually removes the old observation relationship and establishes a new one with a higher frequency.

7.3 Sleepy nodes

Sleepy nodes are devices which occasionally go to a low-power mode by cutting power to unnecessary components to save energy. Some devices cut power only to the radio system while the other components run as usual. At any time, the device could be at a sleeping state or awake. But, in most cases the sleeping time is much larger.

Consequently, communication with sleepy nodes is very problematic, especially if the sleepy node has resources that a client needs from time to time. The major reason for this is that, when a node is in the sleep state, it is disconnected from the network and is unreachable. One solution to resolve this issue efficiently is to use proxy nodes and conditional observations. In the proxy model, all clients get connected to the sleepy node through a proxy. The proxy, then, relays client requests to the server. As soon as the server receives a request, it directly sends back a response to the client via the proxy. However, due to the sleepy nature of the server, the communication is not as simple as this, but the conditional observe mechanism offers an elegant solution to this problem as explained in the Figure 20. In the figure, the red circles are the clients, the gray circle is the server in sleep mode (SleepState = 0), the green circle is the server in awake mode (SleepState = 1), and the middle rectangle is the proxy.

a) A client request arrives at the proxy. The server is sleeping (SleepState = 0). The proxy buffers the

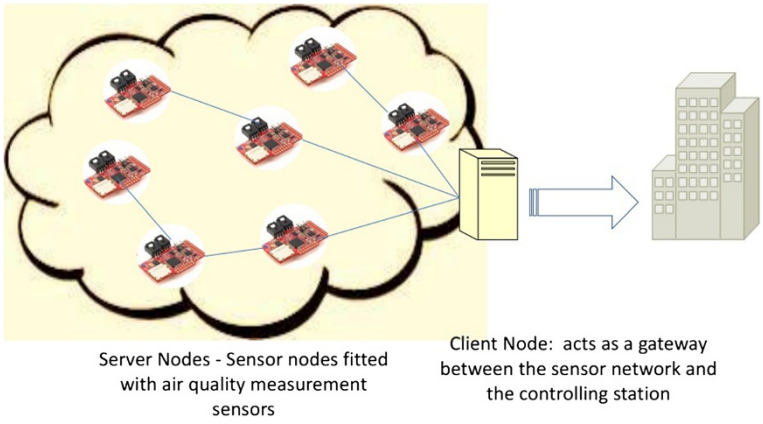

Figure 19 Air quality controlling setup.

request and sends a response to the client telling it to be patient for the actual response. Next the proxy starts to continuously check the server to see if it is awake.

b) As soon as the proxy detects the server is awake (meanwhile retrieving the value requested by the client and delivering the response to the client), it sends a conditional observe request indicating its need to be notified when the server wakes up (SleepState = 1). This can be achieved by using the VALUE = condition type. The server adds the proxy to the observers list.

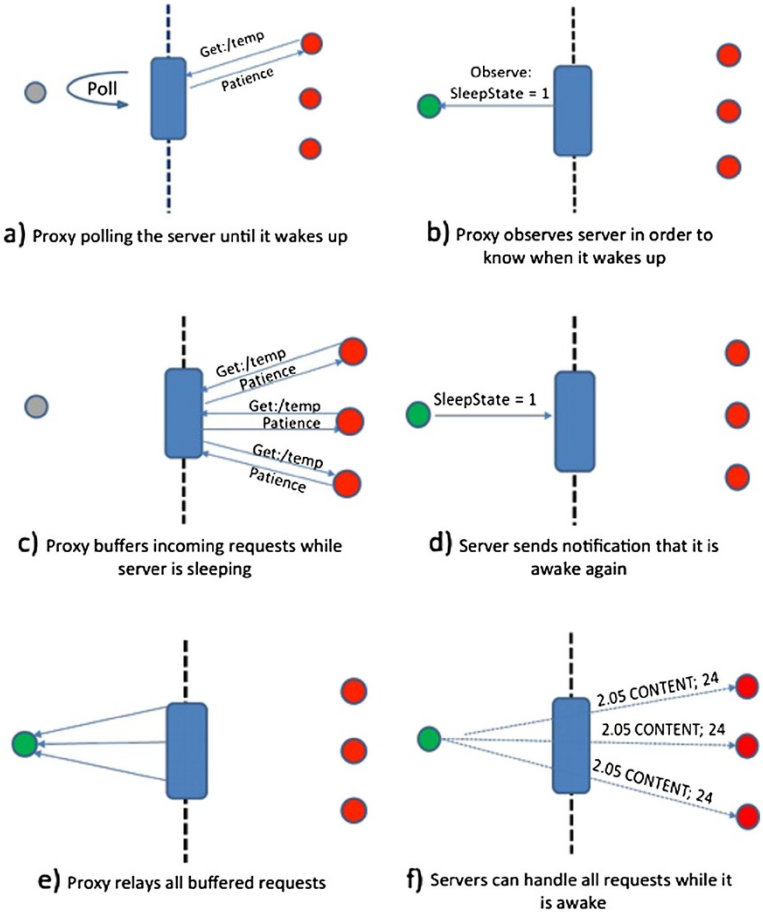

Figure 20 Panels a) to f) illustrate the establishment of communication with sleepy nodes using conditional observation.

c) After some time, the server may go to sleep for a long time (SleepState = 0). While the server is sleeping, clients may send GET requests to the proxy. Since the proxy now knows the server is sleeping, it buffers all requests and sends back patience responses.
d) When the server wakes up, it sends a notification to the proxy indicating its sleep state has changed, back to Awake.
e) When the proxy gets the notification, it sends all buffered requests to the server.
f) Finally, the server sends the responses directly to the clients. As an optimization, the proxy may aggregate similar requests into a single request.
Communication with the sleepy node will be even more efficient if the clients themselves register as observers requesting the server to notify them when a particular criterion is met. In this case, once the server is awake and knows clients' requirement through the proxy, all subsequent notifications will be made directly to the clients whenever the criterion is met. This is done without the involvement of the proxy.

8 Conclusion

In this paper, we presented the concept of conditional observations as an extension to the CoAP protocol in general and the Observe option in particular. We presented comparative results of using normal observation and conditional observation by implementing this functionality on a constrained device. We also presented theoretical evaluations of normal and conditional observation. From both the experimental and theoretical results, it is evident that the conditional observations are very useful extensions to the basic observe behavior, both from an application point of view and from a network efficiency point of view. It enables clients to receive notifications that contain only state changes they are interested in. This has a twofold advantage: an application has the expressiveness to selectively collect data and the data of no interest does not have to travel over the network. The latter advantage will become even more important in larger constrained networks where notifications have to travel over multiple hops. As such, conditional observations can greatly contribute to the reduction of battery consumption and increase of network lifetime. In addition, many scenarios can be thought of that can benefit from this functionality. Finally, our implementation shows the feasibility of implementing this functionality on very constrained devices.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 258885 (SPITFIRE project). We would also like to acknowledge our co-authors of the IETF CoRE conditional observe draft.

Received: 21 February 2013 Accepted: 7 June 2013
Published: 24 June 2013

References

1. Z Shelby, Embedded web services. IEEE Wireless Communications **17**, 52–57 (2010)
2. G Montenegro, N Kushalnagar, D Culler, *RFC4944 - Transmission of IPv6 packets over IEEE 802.15.4 networks, IETF Trust*, 2007
3. J Hui, R Kelsey, P Levis, K Pister, R Struik, JP Vasseur, R Alexander, *RFC6550 - RPL: routing protocol for low power and lossy networks, IETF Trust*, 2012
4. Z Shelby, K Hartke, C Bormann, *Constrained application protocol (CoAP): draft-ietf-core-coap-13, IETF Trust*, 2012
5. Z Shelby, K Hartke, C Bormann, *observing resources in CoAP - draft-ietf-core-observe-07, IETF Trust*, 2012
6. JP Vasseur, A Dunkels, *Connecting Smart Objects with IP: The Next Internet* (Morgan Kaufmann, San Francisco, 2010)
7. Z Shelby, *RFC6690 - Constrained restful environment (CoRE) Link Format, IETF Trust*, 2012
8. AB Roach, *RFC3265 - Session initiation protocol (SIP): specific event notification. IETF Trust*, 2002
9. A Stanford-Clark, HL Truong, *MQTT for Sensor Networks (MQTT-S) Protocol Specification Version 1.1* (IBM Corporation, Armonk, 2008)
10. E Souto, G Guimarães, G Vasconcelos, M Vieira, N Rosa, C Ferraz, J Kelner, *Mires: A Publish/Subscribe Middleware For Sensor Networks* (Springer, London, 2005)
11. S Lai, J Cao, Y Zheng, PS Ware, *A Publish/Subscribe Middleware Supporting Composite Event In Wireless Sensor Network* (IEEE Computer Society, Washington, 2009)
12. European Telecommunications Standards Institute (ETSI), *Machine-to-Machine communications (M2M). ETSI TS 102 690 V1.1.1* (ETSI, France, 2011), p. 10
13. Z Shelby, M Vial, *CoRE Interfaces: draft-shelby-core-interfaces-3, IETF Trust*, 2013
14. A Na, M Priest (eds.), *Sensor Observation Service* (Open Geospatial Consortium Inc, Wayland, 2007)
15. ST Li, J Hoebeke, AJ Jara, *Conditional Observe in CoAP: draft-li-core-conditional-observe-03, IETF Trust*, 2012
16. M Kovatsch, SD Valencia, A low-power CoAP for Contiki, in *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2011)* (IEEE, Amsterdam, 2011), pp. 855–860
17. I Ishaq, J Hoebeke, J Rossey, E De Poorter, I Moerman, P Demeester, Facilitating sensor deployment, discovery and resource access using embedded web services, in *International Workshop on Extending Seamlessly to the Internet of Things (esIoT)*. Palermo, 4–6 July (IEEE, Amsterdam, 2012)

doi:10.1186/1687-1499-2013-177
Cite this article as: Teklemariam *et al.*: Facilitating the creation of IoT applications through conditional observations in CoAP. 2013 **2013**:177.