

core
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2013

Shi Tao. Li
Huawei Technologies
J. Hoebeke
iMinds-IBCN/UGent
A J. Jara
University of Murcia
October 22, 2012

Conditional observe in CoAP
draft-li-core-conditional-observe-03

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. Through the Observe option, clients can observe changes in the state of resources and obtain a current representation of the last resource state. This document defines a new mechanism in CoAP Observe so that a CoAP client can conditionally observe a resource on a CoAP server, only being informed about state changes meeting a specific condition or set of conditions. This offers possibilities to extend network intelligence, enhance scalability, and optimize the lifetime and performance in order to address the requirements from the Constrained Nodes and Networks.

Note

Discussion and suggestions for improvement are requested, and should be sent to core@ietf.org.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Justification	3
1.2. Terminology	4
2. Motivation	4
3. The Condition Option	5
4. Condition Types	8
5. Using the Condition Option	10
6. Cancellation, updating and existence of conditional relationships	11
6.1. Cancellation and updating	11
6.2. Existence	13
7. Discovery	14
8. Examples	15
9. Change log	23
9.1. Changes in v03	23
9.2. Changes in v02	23
10. Security Considerations	23
11. IANA Considerations	23
11.1. Condition option registry	23
11.2. Condition type registry	24
12. Further considerations	25
13. Acknowledgements	25
14. Normative References	25
Appendix A. Alternative approaches	26
A.1. Annex: Cancellation flag	26
A.2. Annex: Logic flag	28
Authors' Addresses	29

1. Introduction

CoAP [I-D.ietf-core-coap] is an Application Protocol for Constrained Nodes/Networks. The observe [I-D.ietf-core-observe] specification describes a protocol design so that a CoAP client and server can use the subject/observer design pattern to establish an observation relationship. When observe is used, the CoAP client will get a notification response whenever the state of the observed resource changed. However, in some scenarios, the CoAP client may only care parts of the state change of the resource, other state changes might be meaningless. This inability to suppress additional notification results in superfluous traffic. This memo defines a new CoAP option "Condition" that can be used to allow the CoAP client to condition the observe request, and only when such condition is met, the CoAP server shall send the notification response with the latest state change. When such a condition fails, the CoAP server does not need to send the notification response.

1.1. Justification

A GET request that includes an Observe Option creates an observation relationship. When a server receives such a request, it first services the request like a GET request without this option and, if the resulting response indicates success, establishes an observation relationship between the client and the target resource. The client is notified of resource state changes by additional responses sent in reply to the GET request to the client.

CoAP is used for Constrained Networks, especially used for transporting sensor data. Different sensor equipments have different properties, e.g. different change rates, data unit, different response time, etc. resulting in varying clients' interests that differ from mere state changes. As such, when a client wants to collect information from a sensor, it does not want to receive useless information. In addition, this would cause the transmission of irrelevant information in an already constrained network.

Consider the following example.

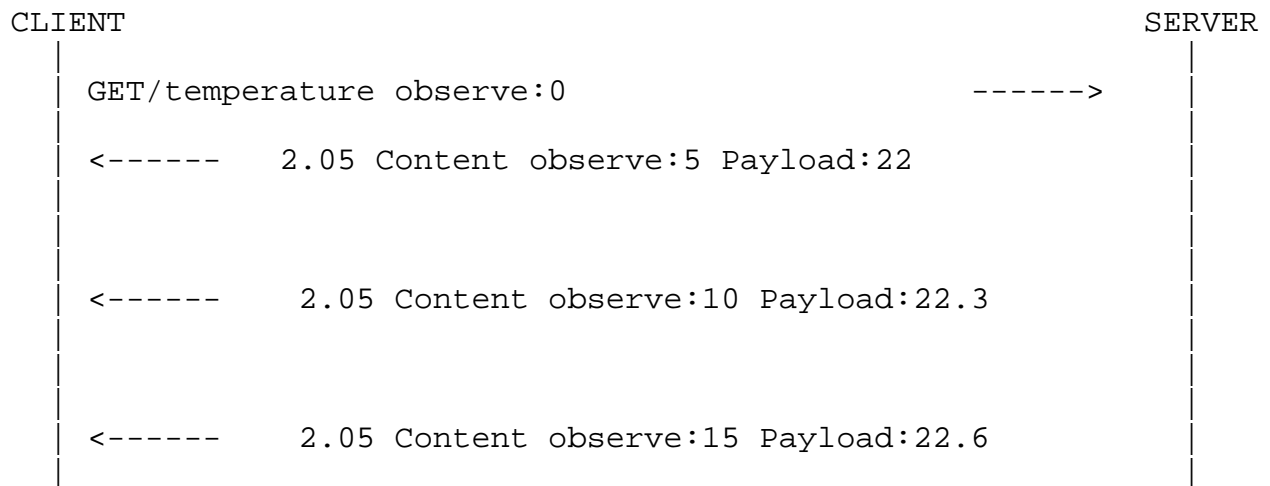


Figure 1: GET request with observe

In this example, the sensor acts as a server, and it collects the resource data every 5 seconds. When the client observes a resource on the server, it will receive a response whenever the server updates the resource, that is to say, mostly every 5 seconds the client will receive a notification response. However, the client might be a quite simple equipment not too sensitive to the resource state change, so it may not want to receive the notification that often. One possible solution could be to change the sensor's parameter, shorten the collecting frequency. However, the sensor should be able to provide services to many other clients, making it hard to find the best configuration that fits all clients' requirements.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Motivation

The CoAP Observe Option gives clients the ability to observe changes in the state of resources. A notification relationship is established and whenever the state of the resource changes, the new representation is pushed to the observer. In many cases, an observer will typically be interested in state changes that satisfy a specific condition. In addition, similar conditional observations will prove useful for many different resources. For example, being informed when the state of a resource exceeds a specific value.

Defining an agreed set of commonly used conditional observations has

a number of advantages. In a well-defined way, clients can observe different resources conditionally. At the same time, these resources can clearly announce how they can be observed, facilitating machine processing of this information. Also, intermediaries can process multiple conditional observations, having as goal to alleviate the load on the constrained network and devices. In the absence of such a set of commonly used conditional observations, where every application developer can specify its own mechanisms, these advantages are lost.

In [I-D.shelby-core-interfaces] a mechanism is described to provide additional information to the Observe Option through the use of query parameters. It is possible to define a fixed set of query parameters to enable conditional observations. However, many more query parameters can be offered by a resource for different purposes. This complicates the automatic processing of conditional observations. To alleviate this problem, this draft proposes to implement frequently occurring conditional observations through the use of a new CoAP Condition Option, having a compact representation and well-defined meaning. For other specific conditional observations, another mechanism such as query parameters can be used to complement the Condition Option.

3. The Condition Option

Type	C/E	Name	Data type	Length	Default
18	E	Condition	uint	1-5 B	(none)

Table 1: Condition Option number

The Condition Option has the option number 18. The last bit indicates it is an elective option and the second to last bit indicates that this is a Proxy Unsafe option (similar to the Observe Option). The delta between the Condition Option and the Observe Option is 12.

The Condition Option can be present in both request and response messages. In both cases, it must be used together with the Observe Option, since it extends the meaning of the Observe Option.

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 the response to the initial GET request message, the Condition Option, together with the Observe Option, indicates 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.

Basically, a similar procedure as described in the observe draft [I-D.ietf-core-observe] is followed, but extended with additional behavior by taking the Condition Option into account. The exact semantics are defined in the sections below.

The size of the Condition Option value is not fixed and can range from 1 to 5 bytes. The value carried is in a specific format that consist of two parts: a mandatory condition header and an optional condition value. The condition header has a fixed length of 1 byte and the condition value, when present, can range from 1 to 4 bytes. The condition header consists of 3 fields: the condition type (TYPE), reliability flag (R) and value type (V).

The Condition Option may occur more than once. If multiple Condition Options are present in a request, their relationship is "AND", meaning that the server will only send a notification when both conditions are fulfilled. In the notifications to such a request, the same Condition Options are present. The Figure 10 presents an example of a multiple condition with "AND" conjunction.

Note that in order to establish an "OR" relationship between conditions, a client simply needs to send separate requests using a different source transport address. The Figure 11 presents an example of OR condition with multiple requests, which are sent in two messages via different ports.

Since this solution could be considered as non-optimal, an alternative solution is proposed for discussion in the Annex "Logic flag", where multiple OR relationships with multiple conditional options can be defined, similar as has been presented for the "AND" conjunction.

The main reason of the current "OR" conjunction mechanism is motivated by making the parsing process as simple as possible.

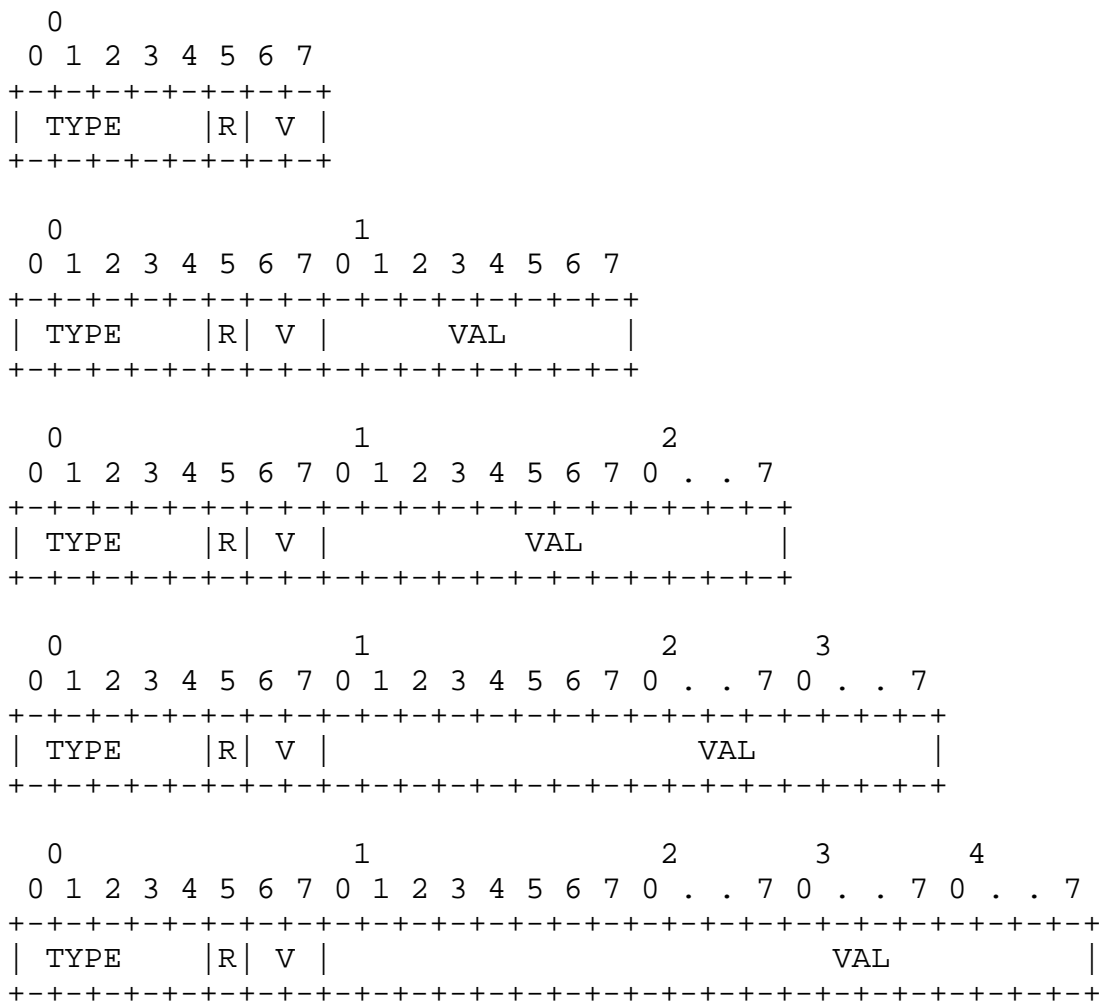


Figure 2: Condition Option value

TYPE: The condition type is a 5 bit integer indicating the type of the condition in the observe request or in a notification. Every value of TYPE represents the type ID of a specific condition type. Every condition type can be complemented by a condition value in the VAL field, further specifying the condition in more detail. Below is the definition of identified condition types.

R: In an observe request, the reliability flag indicates whether notifications for that condition need to be send non-confirmable (0) or confirmable (1). In the initial response, this flag indicates the server’s willingness or ability to send the notifications confirmable or non-confirmable, as requested by the client. In all further notifications, this flag can be changed depending on the server’s decision. In case of a request containing multiple Condition Options, the client must use the same value of the R flag in all Condition Options. If the server receives a request with multiple

Condition Options, which do not all share the same value of the R flag, the server MUST respond with a 4.00 "Bad Request" response code.

V: The value type is a 2 bit field in a request or response that gives the data type of the condition value, when present in the Condition Option. If no condition value is present, this field has no meaning and must be 0. Table 2 gives an overview of all available value types. The Duration data type is defined in Appendix C.2 of [I-D.bormann-coap-misc]. The representation of floating point numbers in a common format that is understandable by constrained devices is outside the scope of this document.

Value type (2 bit)	Id.
Integer	0
Duration (s)	1
Float	2

Table 2: Value types

VAL: The condition value is 1 to 4 byte value of the type indicated by the V flag. The condition value is used to indicate the value that further specifies the condition type (e.g. a threshold). Condition types can require the presence of a condition value. When a condition value is present in an observe request, the same value must be used in the initial response. In all further notifications, the condition value can be left out to reduce the size of the option.

4. Condition Types

Table 3 gives an overview of all currently identified condition types. If supported by the server, different condition types can be combined in a request to express a logical AND relationship. By default, logical OR of condition types is always supported through sending separate requests using a different source transport address.

Condition type (5 bit)	Id.	Condition Value
Cancellation	0	no
Time series	1	no
Minimum response time	2	yes
Maximum response time	3	yes
Step	4	yes
AllValues<	5	yes
AllValues>	6	yes
Value=	7	yes
Value<>	8	yes
Periodic	9	yes

Table 3: Condition types

Time series: This condition indicates that a client wants to receive all state changes of a resource, but that it does not want to receive a notification in case the time since the last notification was sent became greater than the max-age of the resource and the resource did not change during this period. This is a variant of the observe draft [I-D.ietf-core-observe] that deals with eventual consistency, which results in notifications even if the resource did not change in order to ensure the observer has a fresh representation of the resource.

Minimum response time: For this condition, the value specified in the condition value field gives the minimum time in seconds the server should leave between subsequent notifications.

Maximum response time: 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.

Step: For this condition, the value specified in the condition value field gives the minimum state change of a resource (since the last notification) before the server should send a new notification.

AllValues<: This condition indicates that a client is only interested in receiving notifications whenever the state of the resource changes and the value is less than the value specified in the condition value field.

AllValues>: This condition indicates that a client is only interested in receiving notifications whenever the state of the resource changes and the value is greater than the value specified in the condition value field.

Value=: This condition indicates that a client is only interested in receiving notifications whenever the state of the resource changes and the value is equal to the value specified in the condition value field.

Value<>: This condition indicates that a client is only interested in receiving a single notification whenever the state becomes higher or lower than the value specified in the condition value field. 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 notifications is sent whenever a threshold is passed in either direction.

Periodic: This condition indicates the periodic interval in seconds with which new notifications should be sent.

5. Using the Condition Option

Whenever a client wants to initiate a Conditional Observation relationship, it sends a GET request with both an Observe and at least one Condition Option. The Condition Option extends the meaning of the Observe Option by including a condition that describes the resource states the client is interested in. It represents the condition the client wants to apply to the observation relationship.

When a server receives such a request, it first services the request the same way as described in [I-D.ietf-core-observe]. Next, if the server supports the Condition Option, it analyzes the Condition Option to find the condition requested by the client. If the condition is supported, the relationship is stored and the client is informed about the successful establishment of the conditional relationship. This is done by sending a response containing both the Observe and Condition Option, which implies that the client has now been added to the list of observers and will only be informed about state changes or resource states satisfying the condition described in the Condition Option.

Since the Condition Option is elective, an observe request that includes the Condition Option will automatically fall back to a basic observe request if the server does not support the Condition Option. There is no default value for the Condition Option. Also, if the Condition Option is supported, but the requested condition is not supported by the resource, the request will also fall back to a basic observe request, resulting in a response only containing the Observe Option. This implies that the client will now receive notifications as described in [I-D.ietf-core-observe] and that the client itself is responsible for processing the resource state in the notifications in order to identify whether the condition of interest is fulfilled.

Whenever the state of a resource that supports conditional observations changes on the server, the server needs to check the established conditional relationships. Whenever the condition is met, the server sends the notification response to the client that has established the relationship. The response contains both the Observe Option and the Condition Option, where the latter option describes the condition that has been fulfilled. If not met, the server does not send any response to the client.

A client is allowed to use multiple Condition Options in an observe request in order to express a logical AND relationship between different condition types. When a server receives such a request and it does not support composed conditions, the request will also fall back to a basic observe request, resulting in a response only containing the Observe Option. If the server supports this, it will store the relationship and send back a response containing the same multiple Condition Options.

In case a client wants to establish multiple different conditional relationships with the same server, it needs to use a different source transport address for every conditional relationship.

6. Cancellation, updating and existence of conditional relationships

6.1. Cancellation and updating

In case a client wants to cancel an existing conditional relationship, it has to perform a GET request without Observe and Condition Option using the same source transport address used to establish the conditional relationship (i.e. source transport address of the original request). Upon reception of such a GET request, the server will remove the client from the list of conditional observers for that resource.

Alternatively the client can also send a confirmable request

containing a Condition Option with condition type `_Cancellation_`. The source transport address used by the client uniquely identifies the conditional relationship the client has with the server. Upon reception of such a message, the server knows that the client wants to terminate the relationship it has established.

This cancellation mechanism implies that whenever a client wants to establish multiple different conditional relationships with the same server, it needs to use a different source transport address for every conditional relationship.

When a client has established a conditional relationship and it sends a new conditional observe request using the same source transport address, the existing relationship is removed and the new relationship established. This way, a client is able to update existing relationships.

Apart from the cancellation through sending a GET request without Observe and Condition Option, a conditional relationship can also be cancelled by sending a RST message in response of a confirmable notification. When a client rejects a non-confirmable notification with a RST, the server can remove the client from the list of observers interested in the specific condition of the resource in case the server maintains state information about non-confirmable notifications.

Next to this, if the server is for whatever reason not able to further fulfill the conditional relationship of a client, the server can also send a confirmable notification containing a Condition Option with condition type `'Cancellation'`. The source transport address used by the client uniquely identifies a conditional relationship with a server. As such, upon reception of such a message, the client knows that the relationship it has established with that server is terminated.

Finally, when a server sends confirmable notifications that are not acknowledged by the observer, the server may terminate the relationship after X unsuccessful notifications (X implementation dependent).

Note: The possibilities to establish a Cancellation flag have also been evaluated, see Annex "Cancellation flag". This has been discarded since, as it is too complex for processing, when multiple conditions are defined.

6.2. Existence

A client has the possibility to establish and remove conditional relationships. A server can also inform a client about the removal of a conditional relationship. Next to this, there is the issue of how long the relationship is guaranteed to exist in the absence of any explicit removal from either the client or server side (e.g. a client that wants to maintain the relationship for a very long time) or in the absence of frequent notifications. To this end, a mechanism is needed for a client to know whether it is still on the list of observers and for a server to know whether a client is still an observer.

In order for a client to know whether it is still on the list of observers after a long period without notifications or without confirmable notifications, the server can use the Pledge Option, as defined in [I-D.bormann-coap-misc]. By adding this option to its notifications, the server indicates how long it minimally promises to maintain that specific conditional relationship. In case no new notifications or non-confirmable notifications are sent and the duration indicated in the Pledge Option is to expire, the client must renew the relationship by resending the same request, preferable as a confirmable message.

In case the duration indicated in the Pledge Option expires at the server side and the client did not renew the relationship, the server must remove the relationship by sending an explicit cancellation message (a confirmable notification to the client's source transport address containing a Condition Option with condition type 'Cancellation'). As such, the use of the Pledge Option extends the establishment and removal mechanism with a server-initiated mechanism to realize intermediate refreshments of the relationship.

The second mechanism, a server to know whether a client is still an observer, is realized by adding a Keep-Alive Option to the observe request. The size of the Keep-Alive Option value is 1 byte and represents a duration in seconds, using the Duration data type as defined in Appendix C.2 of [I-D.bormann-coap-misc].

Type	C/E	Name	Format	Length	Default
30	E	Keep-alive	Duration in s	1 B	(none)

Table 4: Keep-alive Option number

The Keep-Alive Option has the option number 30, meaning that it is an elective, Proxy Safe option, but not a cache key.

When the client adds the Keep-Alive option to its conditional observe request, it requests the server to confirm that the relationship is still alive every time the duration expires and no notifications or only non-confirmable notifications have been sent during that period. If the option is supported by the server, the option is added to the response. Every time the duration expires and no notifications or only non-confirmable notifications have been sent, the server sends a confirmable notification to the client with an empty payload (since the condition is not fulfilled). As such, the use of the Keep-alive Option extends the establishment and removal mechanism with a client-initiated mechanism to realize intermediate refreshments of the relationship.

The Pledge Option allows a server to request a client to confirm its interest and the Keep-Alive option allows a client to request a server to confirm whether it is still an observer. In case neither of the two options are supported, the only way for the client to ensure the relationship is still existing in the absence of incoming notifications is to periodically reestablish the relationship and the only way for the server to ensure the client is still interested is to send a confirmable notification from time to time.

7. Discovery

The Condition Option enables the establishment of well-defined set of conditional observations. It is equally important for a resource to be able to announce in a well-defined way which conditional observations it supports. Clients can then discover these capabilities and process them automatically.

In [I-D.ietf-core-observe], the "obs" attribute is introduced as a target attribute. It is used as a flag without any value, indicating that the resource is observable. In order to describe the conditional observe capabilities of a resource, a value is added to the obs attribute. To describe which of the 2^5 possible condition types a resource supports, a 32-bit value is used where a bit-value of 1 at position X (starting from 0 and from right to left) indicates that the condition type X is supported. As such, by a client can discover the supported observe capabilities by parsing the value of the "obs" attribute. In case no value is present, the "obs" attribute preserves its original meaning.

8. Examples

This section gives a number of short examples with message flows to illustrate the use of Condition Option in a observe GET request. Note that, in order to keep the figures readable, the Condition Option that is included in every notification is not shown.

The first example (Figure 3) shows that the client sets the Condition Option to Type: 1 - No Value (1/-). The condition is Time Series, meaning that the client wants to receive all state changes. In case the state of the resource does not change, no notifications are sent, also not in case max-age expires. We assume a max-age value of 60 seconds.

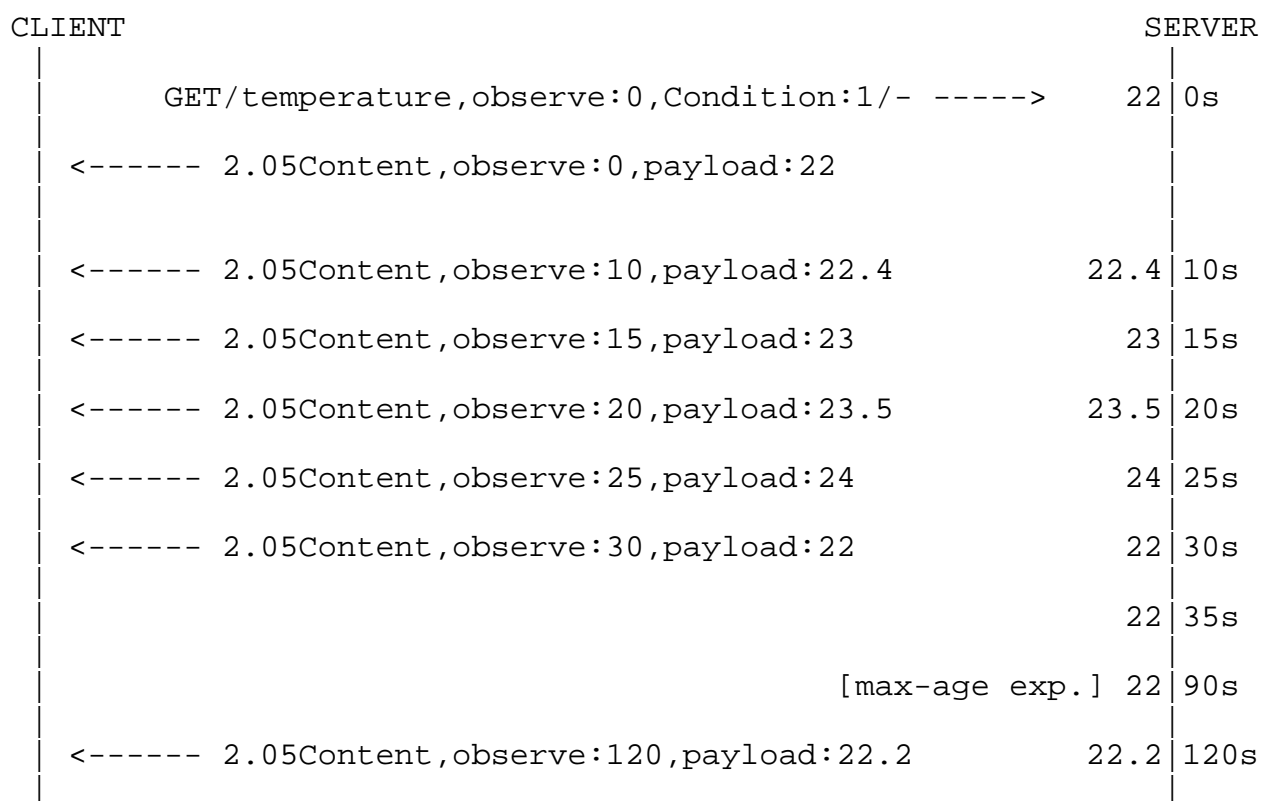


Figure 3: Condition Option with value 1/- (Time Series)

The following example (Figure 4) shows that the client sets the Condition Option to Type: 2 - Value: 10 (2/10), This means that the server shall wait at least 10s between sending notification responses, indicating changes in the state of the resource, to the client.

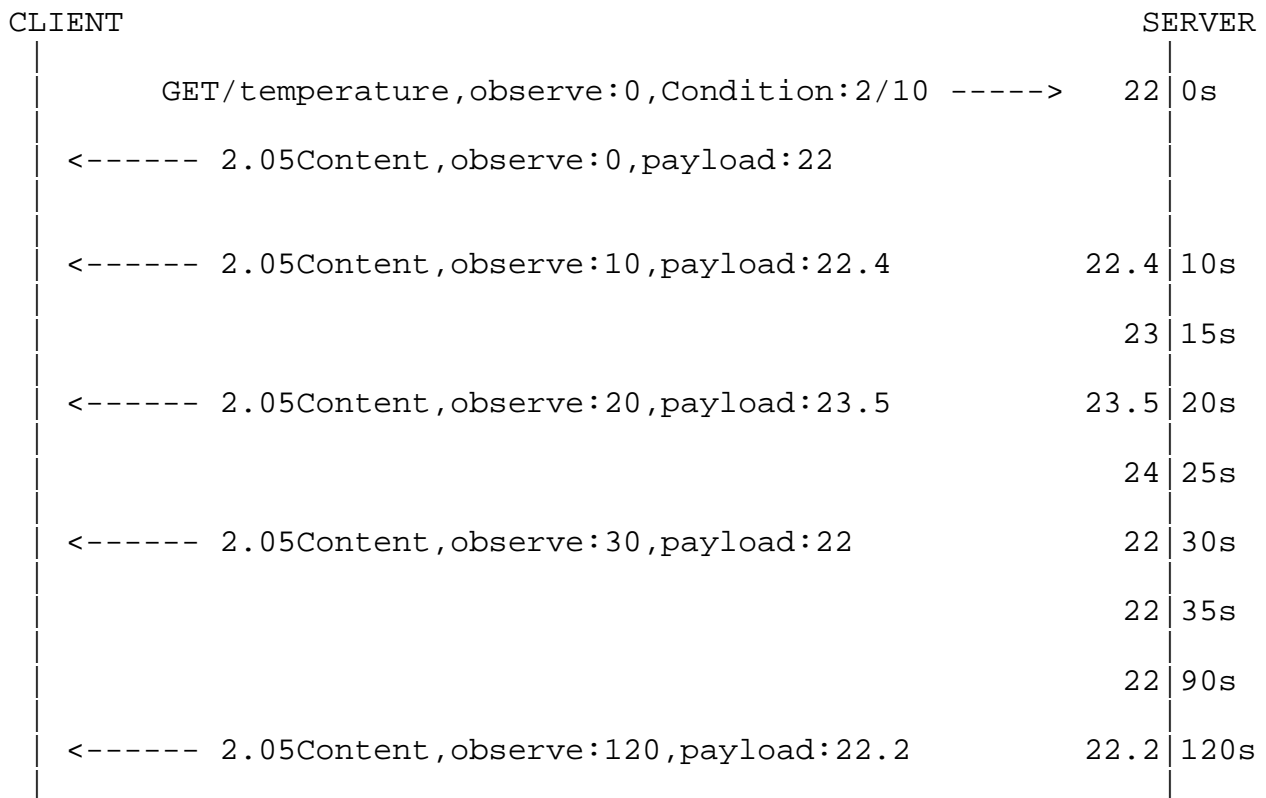


Figure 4: Condition Option with value 2/10 (Minimum Response Time)

The next example (Figure 5) shows that the client sets the Condition Option to Type: 3 - Value: 60 (3/60). The server will send notifications upon every state change, but will leave maximally 60s between subsequent notifications, even if they do not incur a state change.

CLIENT		SERVER
	GET/temperature,observe:0,Condition:3/60 ----->	22 0s
<-----	2.05Content,observe:0,payload:22	
<-----	2.05Content,observe:10,payload:22.4	22.4 10s
<-----	2.05Content,observe:15,payload:23	23 15s
<-----	2.05Content,observe:20,payload:23.5	23.5 20s
<-----	2.05Content,observe:25,payload:24	24 25s
<-----	2.05Content,observe:30,payload:22	22 30s
		22 35s
<-----	2.05Content,observe:90,payload:22	22 90s
<-----	2.05Content,observe:120,payload:22.2	22.2 120s

Figure 5: Condition Option with value 3/60 (Maximum Response Time)

Figure 6 shows a client setting the Condition Option to Type: 4 - Value: 1 (4/1). The server will now send notifications every time the change in state of the resource is at least 1.

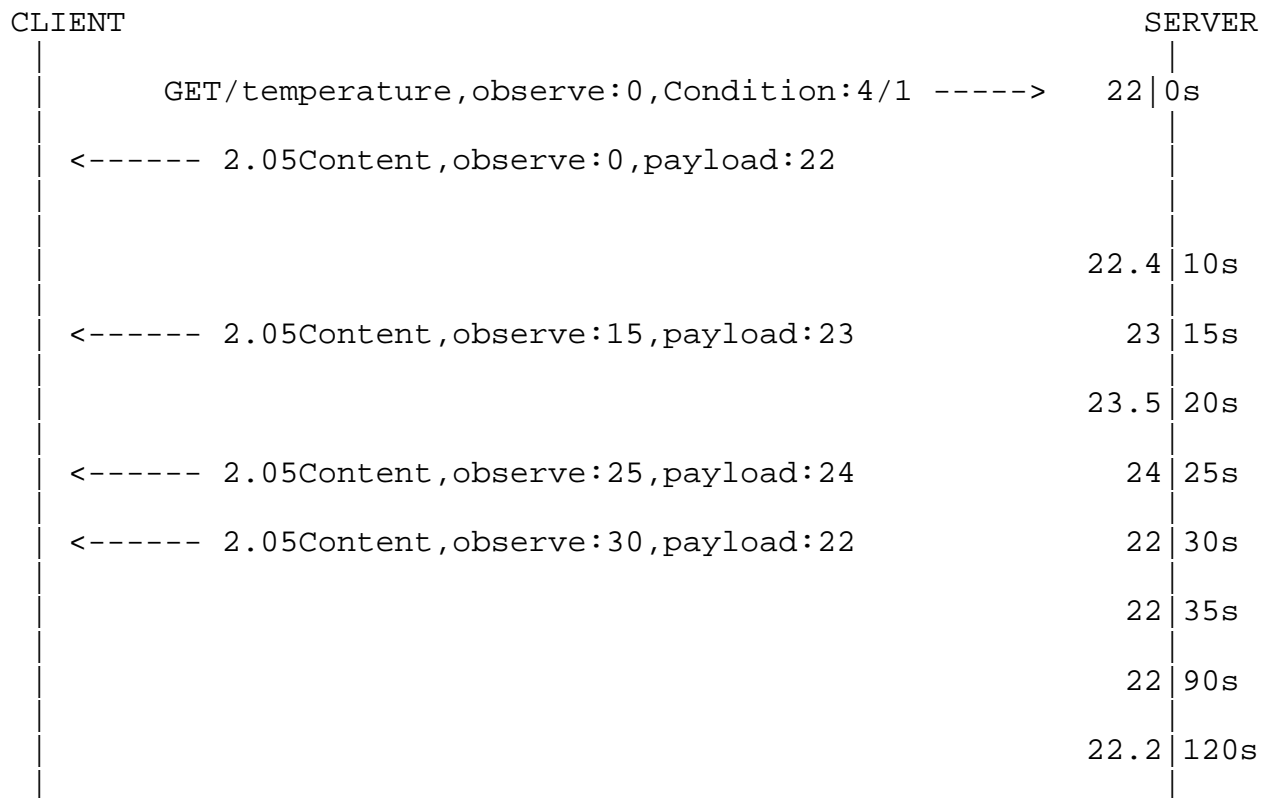


Figure 6: Condition Option with value 4/1 (Step)

The example in Figure 7 shows that the client sets the Condition Option to Type: 6 - Value: 23 (6/23). The server will send notifications to the client only if the resource value is bigger than 23.

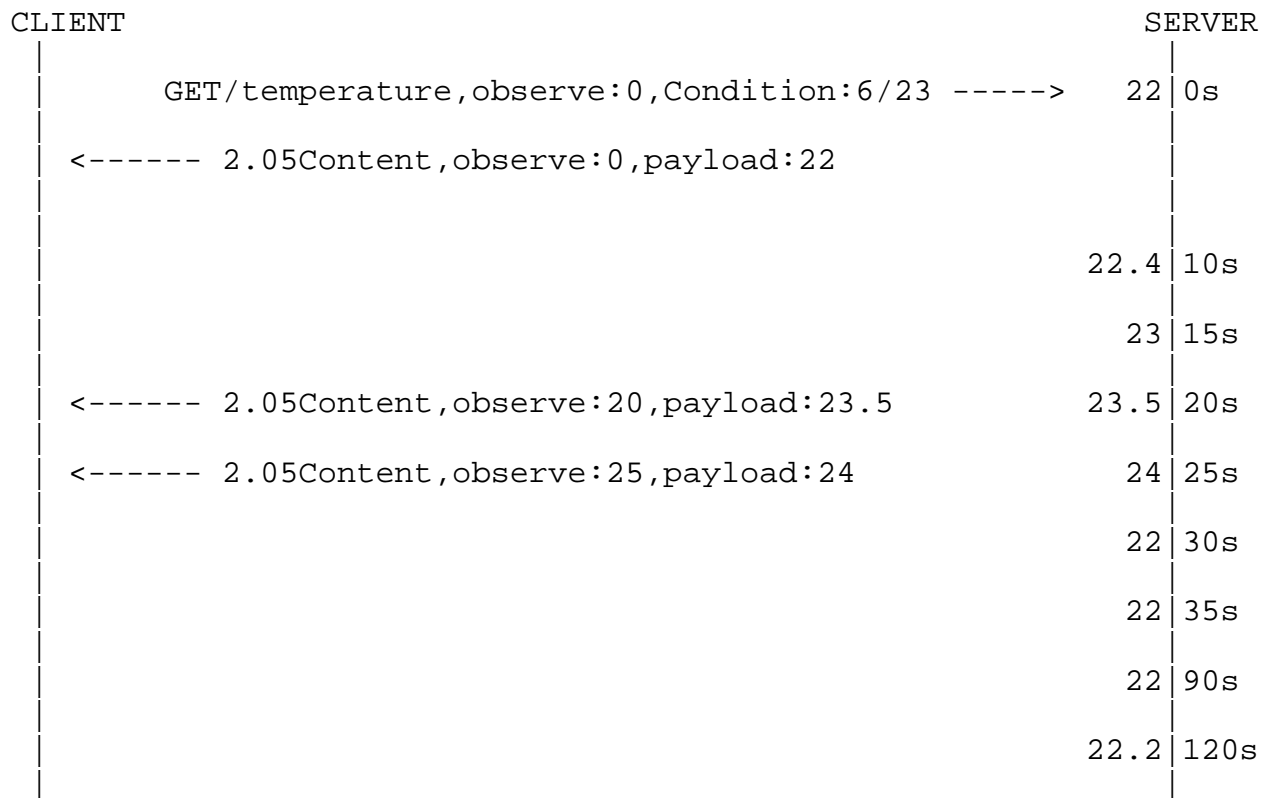


Figure 7: Condition Option with value 6/23 (AllValues>)

Figure 8 is an example of a client setting the Condition Option to Type: 8 - Value: 23 (8/23). The server will send a single notification whenever the state becomes higher or lower than 23.

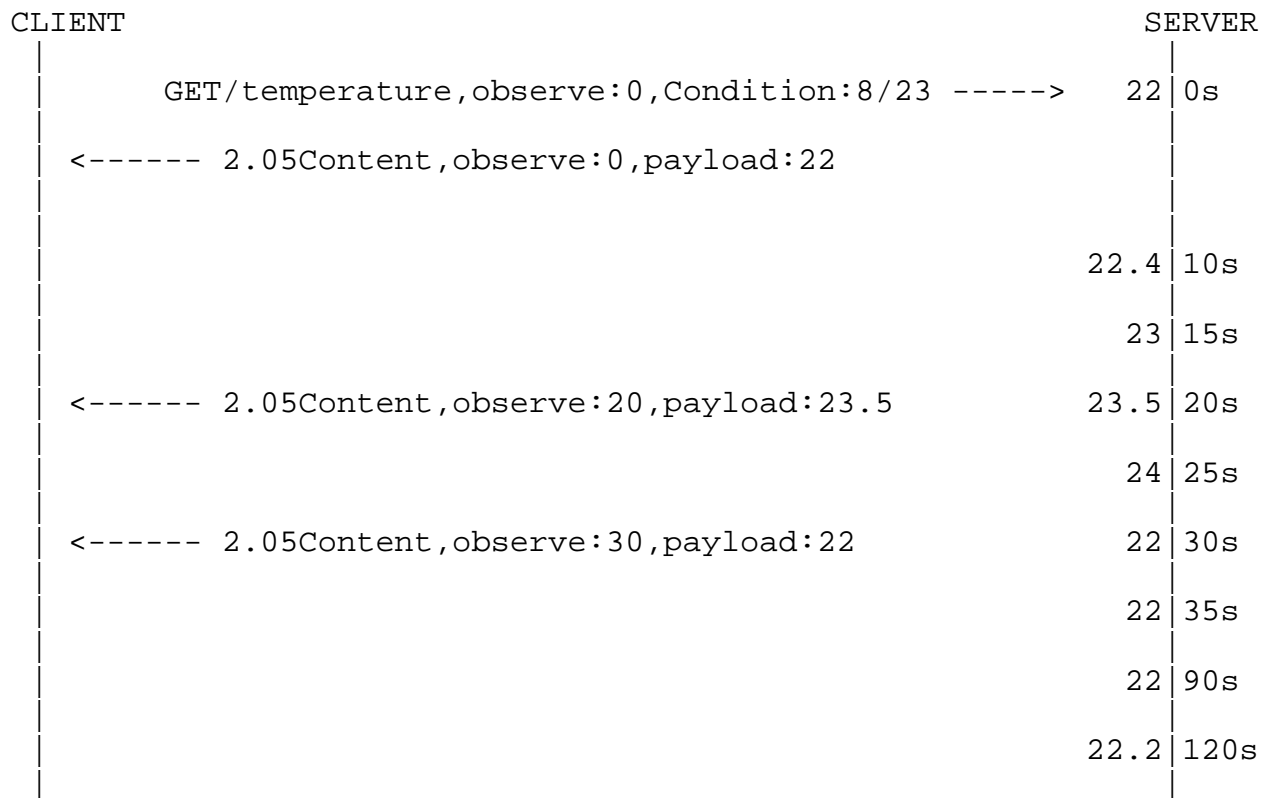


Figure 8: Condition Option with value 8/23 (Values<>)

Figure 9 is an example of a client setting the Condition Option to Type: 9 - Value: 30 (9/30). The server will send notifications every 30 seconds, independent whether the resource has changed or not.

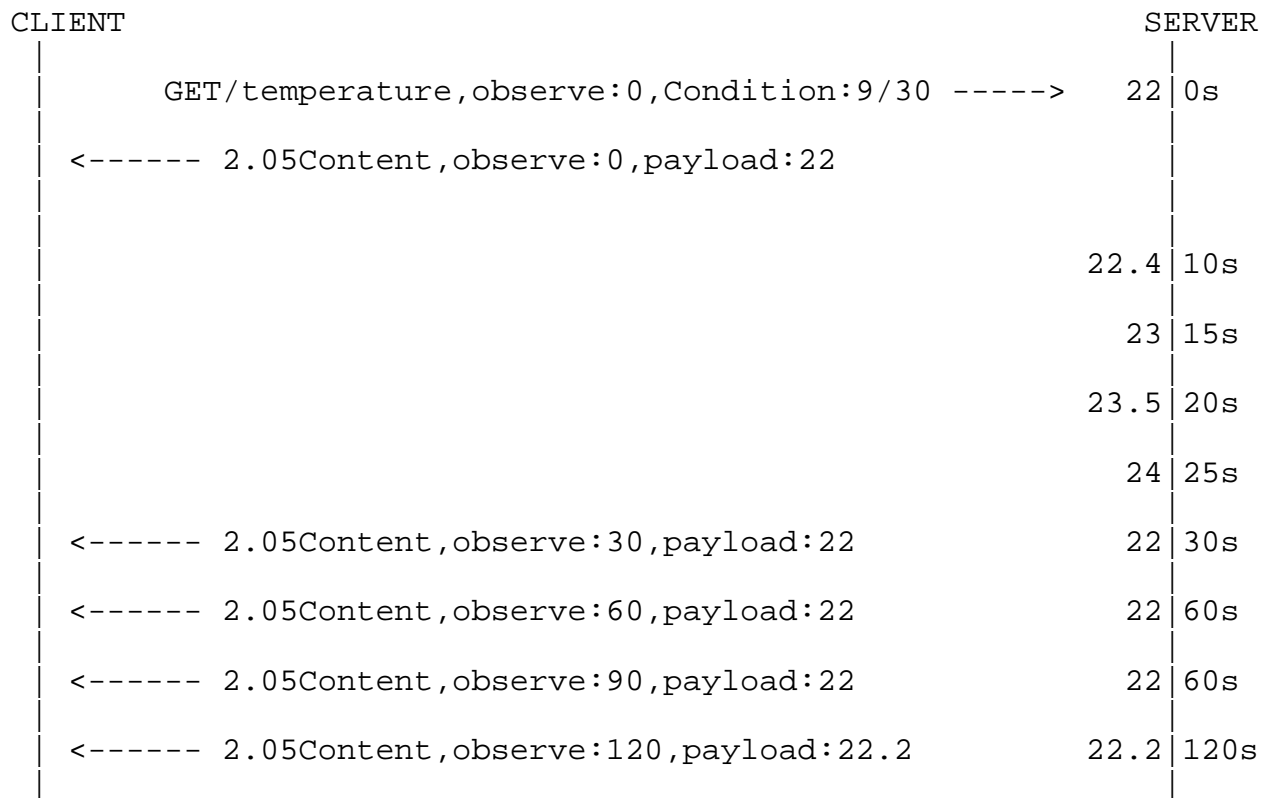


Figure 9: Condition Option with value 9/30 (Periodic)

In the following examples, we illustrate the combination of different conditions. The example in Figure 10 shows the client adds two range Condition Options in the request, one set to 6/5, another one set to 5/15. It means that the range is within 5 and 15, i.e. value > 5 AND value < 15. Since it is an AND condition, the two conditions can be specified in the same observe with multiple options.

CLIENT	SERVER
<pre>GET/temperature, observe:0,Condition:6/5, Condition:5/15</pre>	<pre>-----></pre>
<pre><----- 2.05Content,observe:0,payload:4</pre>	<pre>4 0s</pre>
	<pre>3 5s</pre>
	<pre>3 10s</pre>
<pre><----- 2.05Content,observe:15,payload:12</pre>	<pre>12 15s</pre>
	<pre>16 20s</pre>
<pre><----- 2.05Content,observe:25,payload:14</pre>	<pre>14 25s</pre>

Figure 10: Two Condition Options to define in-side a range option
6/5 AND 5/15 (AllValues> AND AllValues<)

The last example (Figure 11) shows the client adds two Observe request messages to build a range, one sets to 6/22, another one sets to 5/16. It means that the range is out of range between 16 and 22, i.e. value > 22 OR value < 16. This requires two messages, since it is the OR option, which is defined with multiple observe messages. An example of the application for this option can be found in the implementation of the conditional observer [SENSORS].

CLIENT	SERVER
<pre>GET/temperature, observe:0,Condition:6/22 (PORT X) ---> GET/temperature, observe:0,Condition:5/16 (PORT Y) ---></pre>	<pre>18</pre>
<pre><----- 2.05Content,observe:10,payload:22,5</pre>	<pre>22.5</pre>
<pre><----- 2.05Content,observe:20,payload:23.2</pre>	<pre>23.2</pre>
	<pre>19</pre>
<pre><----- 2.05Content,observe:35,payload:15</pre>	<pre>15</pre>

Figure 11: Two Observe requests to define out-side a range
6/22 OR 5/15 (Allvalues> OR Allvalues<)

Further, in [CPSCOM], an evaluation can be found regarding the feasibility of implementing conditional observations on real constrained devices, together with a basic performance comparison between conditional observe (server-filtering) and normal observe in combination with client-side filtering.

9. Change log

9.1. Changes in v03

- o Examples for most condition types
- o Update the option number according to the new numbering scheme
- o Added reference to paper validating implementation on constrained device

9.2. Changes in v02

- o Restructured entire document
- o Detailed description of the Condition Option and updated format of the Condition Option value
- o Added more Condition Types
- o New section on cancellation, updating and existence of conditional relationships
- o New section on discovery

10. Security Considerations

As the Condition Option is used together with the Observe option, when it is used it must follow the security considerations as described in Observe draft[I-D.ietf-core-observe].

11. IANA Considerations

11.1. Condition option registry

This draft adds the following option numbers to the CoAP Option Numbers registry of [I-D.ietf-core-coap]

Number	Name	Reference
26	Condition	[RFCXXXX]
28	Keep-alive	[RFCXXXX]

Table 3: Condition and Keep-alive Option number

11.2. Condition type registry

The Condition types defined in this draft are identified by a string, such as "Step". In order to minimize the overhead of using these condition types, this document defines a registry for the condition types to be used in CoAP and assigns each a numeric identifier.

Each entry in the registry must include the condition type registered with IANA, the numeric identifier in the range 0-31 to be used for that condition type in CoAP, and a reference to a document defining the usage of that condition type.

Initial entries in this registry are as follows:

Condition type	Id.	Reference
Cancellation	0	[RFCXXXX]
Time series	1	[RFCXXXX]
Minimum response time	2	[RFCXXXX]
Maximum response time	3	[RFCXXXX]
Step	4	[RFCXXXX]
AllValues<	5	[RFCXXXX]
AllValues>	6	[RFCXXXX]
Value=	7	[RFCXXXX]
Threshold	8	[RFCXXXX]
Periodic	9	[RFCXXXX]

Table 5: Condition Option type

12. Further considerations

Intermediaries, caching, retransmissions

13. Acknowledgements

Thanks to the IoT6 European Project (STREP) of the 7th Framework Program (Grant 288445).

14. Normative References

[CPSCOM] Ketema, G., Hoebeke, J., Moerman, I., Demeester, P., Li, Shi Tao., and A. Jara, "Efficiently observing Internet of Things Resources", The 2012 IEEE International Conference on Cyber, Physical and Social Computing November 20-23, 2012, BesanAS.on, France, Novemer 2012.

[I-D.bormann-coap-misc]

Bormann, C. and K. Hartke, "Miscellaneous additions to

CoAP", draft-bormann-coap-misc-13 (work in progress),
March 2012.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank,
"Constrained Application Protocol (CoAP)",
draft-ietf-core-coap-12 (work in progress), October 2012.

[I-D.ietf-core-link-format]

Shelby, Z., "CoRE Link Format",
draft-ietf-core-link-format-14 (work in progress),
June 2012.

[I-D.ietf-core-observe]

Hartke, K., "Observing Resources in CoAP",
draft-ietf-core-observe-06 (work in progress),
September 2012.

[I-D.shelby-core-interfaces]

Shelby, Z., "CoRE Interfaces",
draft-shelby-core-interfaces-01 (work in progress),
January 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.

[SENSORS] Castro, M., Jara, A., and A. Skarmeta, "Architecture for
Improving Terrestrial Logistics Based on the Web of
Things", Sensors 12, no. 5, 6538-6575, 2012, May 2012.

Appendix A. Alternative approaches

In this appendix, we include some alternative solutions the authors have discussed regarding cancellation of relationships and the logical combination of different relationships. The mechanisms described here allow more flexibility, but introduce additional (undesired?) complexity. We put their description in this appendix to trigger further discussion and provide more background.

A.1. Annex: Cancellation flag

An alternative way to allow the explicit establishment and removal of conditional relationships and to allow the establishment of multiple conditional relationships to the same server using the same source transport address, is through the introduction of a 1 bit cancellation flag (C) as part of the 1 byte condition header. The following paragraphs describe how this flag would change this

behavior.

C: The cancellation flag, when used in an observe request, indicates whether the client wants to establish a conditional observation relationship (0) or cancel an existing conditional relationship (1). In the initial response, the server uses the same value as in the request. In all further notifications, this flag has no meaning and must be 0. In case of a request containing multiple Condition Options, the client must use the same value of the C flag in all Condition Options. The cancellation flag allows a client to establish multiple conditional observation relationships and remove individual relationships from the same address and port.

When using this cancellation flag, a client is able to establish multiple conditional relationships using the same source transport address. Different from [I-D.ietf-core-observe] a GET without observe issued by a client, will not result in the the removal of established conditional relationships. Instead the client has the possibility to explicitly terminate any established conditional relationship by sending the same observe request, but with the Condition Option having the C flag set in order to trigger the cancellation of the request. This way, the same end point can manage multiple conditional observation relationships without the risk of accidentally removing them.

When a server receives a cancellation request, it removes the relationship indicated by the Condition Option and sends back a response containing both the Observe and the Condition Option with the cancellation flag set to 1.

In case a client wants to terminate all existing conditional observation relationships with a server, it should send a request with Condition Option, where the Condition Type is set to the reserved value 0 and the cancellation flag to 1. Upon reception of this message, the server removes all existing relationships and sends back a response containing the same Condition Type.

In case a client has established a conditional relationship that is the result of a request with multiple Condition Options, the client can cancel this relationship by sending the same request, but now with all cancellation flags set to 1.

If the server is for whatever reason not able to further fulfill the conditional relationship of a client, the server can also send a confirmable notification containing the Condition Option with the C flag set to 1 in order to terminate the observation relationship.

A.2. Annex: Logic flag

Different condition types can be combined in a request to express a logical AND relationship. By default, logical OR of condition types is always supported through sending separate requests. In order to express an out-of-range condition (notification when value is lower than X OR higher than Y), two separate conditional observe requests have to be sent. Through the introduction of a 1 bit logic flag (L) as part of the 1 byte condition header, this can be avoided.

L: The logic flag in a Condition Option indicates how the condition should be combined logically with the condition in the next Condition Option. A value of 0 means AND and a value of 1 means OR. The flag has no meaning if the Condition Option is the only or last Condition Option in the request. Through the use of the L flag it is possible to logically combine different conditions in a single request (e.g. C1 AND C2 OR C3 AND C4). As a drawback, when used, it complicates processing at the server side.

This example (Figure 12) shows the same example from the (Figure 11) but with this alternative. This example presents as the client adds two Observe options to build a range, one set to 6/22, another one set to 5/16. It means that the range is out of range between 16 and 22, i.e. value > 22 OR value < 16. The first option is defined by default with the Logic flag equal to 0. The second option is defined with the the Logic flag equal to 1, since it is the OR option. Note as it has been simplified from two messages to only one message with two options, and the most important, it has avoided the management of multiple ports.

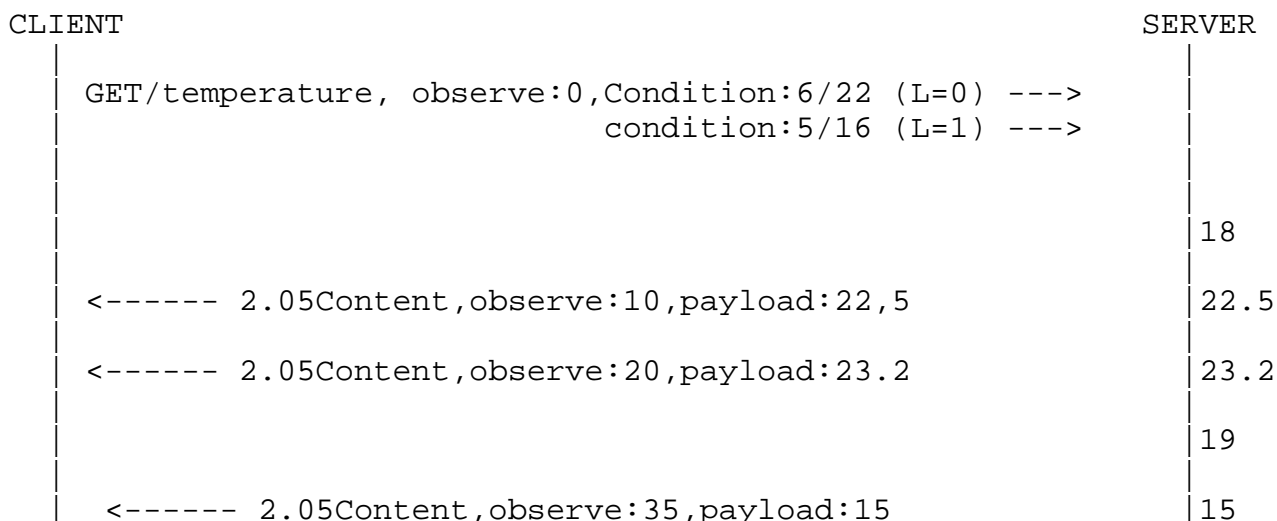


Figure 12: Two Observe options with Logic flag to define out-side a range 6/22 OR 5/15.

Authors' Addresses

Shitao Li
Huawei Technologies
Huawei Base
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Phone: +86-25-56624157
Email: lishitao@huawei.com

Jeroen Hoebeke
iMinds-IBCN/UGent
Department of Information Technology
Internet Based Communication Networks and Services (IBCN)
Ghent University - iMinds
Gaston Crommenlaan 8 bus 201
Ghent B-9050
Belgium

Phone: +32-9-3314954
Email: jeroen.hoebeke@intec.ugent.be

Antonio J. Jara
University of Murcia
Department of Information Technology and Communications
Computer Science Faculty
Campus de Espinardo
Murcia ES-30100
Spain

Phone: +34-868-88-8771
Email: jara@um.es