CoRE
Internet Draft
Intended status: Informational
Expires: October 2016

A. Bhattacharyya
S. Bandyopadhyay
A. Pal
T. Bose
Tata Consultancy Services Ltd.

April 4, 2016

CoAP option for no server-response
draft-tcs-coap-no-response-option-15

Abstract

There can be M2M scenarios where responses from a server against requests from client are redundant. This kind of open-loop exchange (with no response path from the server to the client) may be desired to minimize resource consumption in constrained systems while updating a bulk of resources simultaneously, or updating a resource with a very high frequency. CoAP already provides Non-confirmable (NON) messages that are not acknowledged by the recipient. However, the request/response semantics still require the server to respond with a status code indicating "the result of the attempt to understand and satisfy the request".

This specification introduces a CoAP option called 'No-Response'. Using this option the client can explicitly tell the server to suppress all responses against the particular request. This option also provides granular control to enable suppression of a particular class of response or a combination of response-classes. This option may be effective for both unicast and multicast requests. This document also discusses a few exemplary applications which benefit from this option.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on October 4, 2016.

Copyright Notice

Table of Contents

1. Introduction

This specification defines a new option for Constrained Application Protocol (CoAP) [RFC7252] called 'No-Response'. This option enables clients to explicitly express their disinterests in receiving responses back from the server. The disinterest can be expressed at the granularity of response classes (e.g., 2.xx or the combination of 2.xx and 5.xx). By default this option indicates interest in all response classes.

Along with the technical details this document presents some practical application scenarios which bring out the usefulness of this option.

Wherever, in this document, it is mentioned that a request from a client is with No-Response the intended meaning is that the client expresses its disinterest for all or some selected classes of responses.

1.1. Potential Benefits

Use of No-Response option should be driven by typical application requirement and, particularly, characteristics of the information to be updated. If this option is opportunistically used in a fitting M2M application then the concerned system may benefit in the following aspects (however, it is to be noted, this option is elective and servers can simply ignore the preference expressed by the client):

* Reduction in network congestion due to effective reduction of the overall traffic.

* Reduction in server-side load by relieving the server from responding to each request when not necessary.

* Reduction in battery consumption at the constrained end-point(s).

* Reduction in overall communication cost.

1.2. Terminology

The terms used in this document are in conformance with those defined in [RFC7252].

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 RFC-2119.

2. Option Definition

The properties of No-Response option are given in Table 1.

Number	C	U	N	R	Name	Format	Length	Default
284			X		No-Response	uint	0-1	0

Table 1: Option Properties

This option is a request option. It is Elective and Non-Repeatable.

Note: Since CoAP maintains a clear separation between the request/response and the message sub-layer, this option does not have any dependency on the type of message (Confirmable/Non-confirmable). So, even the absence of message sub-layer (ex. CoAP over TCP) should have no effect on the interpretation of this option. However, considering the CoAP-over-UDP scenario, NON type of messages are best fitting with this option, considering the expected benefits out of it. Using No-Response with NON messages gets rid of any kind of reverse traffic and the interaction becomes completely open-loop.

Using this option with CON type of requests may not serve the desired purpose if piggybacked responses are triggered. But, in case the server responds with a separate response (which, perhaps, the client does not care about) then this option can be useful. Suppressing the separate response reduces traffic by one additional CoAP message in this case.

This option contains values to indicate disinterest in all or a particular class or combination of classes of responses as described in the next sub-section.

2.1. Granular Control over Response Suppression

This option enables granular control over response suppression by allowing the client to express its disinterest in a typical class or combination of classes of responses. For example, a client may explicitly tell the receiver that no response is required unless something 'bad' happens and a response of class 4.xx or 5.xx is to be fed back to the client. No response of the class 2.xx is required in such case.

Note: Section 3.7 of [RFC7390] describes a scheme where a server in the multicast group may decide on its own to suppress responses for group communication with granular control. The client does not have any knowledge about that. However, on the other hand, the 'No-Response' option enables the clients to explicitly inform the servers about its disinterest in responses. Such explicit control on the client side may be helpful for debugging network resources. An example scenario is described in Section 3.2.

The option value is defined as a bit-map (Table 2) to achieve granular suppression. Its length can be 0 byte (empty value) or 1 byte.

Value	Binary Representation	Description
0	<empty>	Interested in all responses.
2	00000010	Suppress 2.xx responses.
8	00001000	Suppress 4.xx responses.
16	00010000	Suppress 5.xx responses.
127	01111111	Suppress all responses.

Table 2: Option values

The conventions used in deciding the option values are:

1. To suppress an individual class: Set bit number (n-1) starting from the LSB (bit number 0) to suppress all responses belonging to class n.xx. So,

option value to suppress n.xx class = 2**(n-1).

2. To suppress combination of classes: Set each corresponding bit according to point 1 above. Example: The option value will be 18 (binary: 00010010) to suppress both 2.xx and 5.xx responses. This is essentially bitwise OR of the corresponding individual values for suppressing 2.xx and 5.xx. At present the "CoAP Response Codes" registry (Ref. Section 12.1.2 of [RFC7252]) defines only 2.xx, 4.xx and 5.xx responses. So, an option value of 26 (binary: 00011010) will effectively suppress all currently defined response codes.

3. To suppress all possible responses: The maximum reserved response code for CoAP is 7.31 (Ref. Section 12.1 of [RFC7252]). So, setting bit positions 0-6 will suppress all responses according to the combination operation defined in point 2 above. Hence, the value to block all present and possible future responses is 127 (binary: 01111111).

Note: When No-Response is used with value 127 in a request the client end-point SHOULD cease listening to response(s) against the particular request. On the other hand, showing interest in at least one class of response means that the client end-point can no longer completely cease listening activity and must be configured to listen up to some application specific time-out period for the particular request. The client end-point never knows whether the present request will be a success or a failure. Thus, for example, if the client decides to open up the response for errors (4.xx and 5.xx) then it has to wait for the entire time-out period even for the instances where the request is successful (and the server is not supposed to send back a response). A point to be noted in this context is that there may be situations when the response on errors might get lost. In such a situation the client would wait up to the time-out period but will not receive any response. But this should not lead to the impression to the client that the request was necessarily successful. In other words, in this case the client cannot distinguish between response suppression and message loss. The application designer needs to tackle such situation. For example, while performing frequent updates, the client may strategically

interweave requests without No-Response option into a series of requests with No-Response to check time to time if things are fine at the server end and the server is actively responding.

2.2. Method-specific Applicability Consideration

The following table provides a ready-reference on the possible applicability of this option for all the four REST methods. This table is prepared in view of the type of possible interactions foreseen at time of preparing this specification. Capitalization of key words like "SHOULD NOT", etc. have not been deliberately used in this section as this is a purely a suggestive table.

Method Name	Remarks on applicability
GET	This should not be used with conventional GET request when the client requests the contents of a resource. However, this option may be useful for exceptional cases where GET requests has side effects. For instance, the proactive 'cancellation' procedure for observing request [RFC7641] requires a client to issue a GET request with Observe option set to 1 ('deregister'). In case it is more efficient to use this deregistration instead of reactive cancellation (rejecting the next notification with RST), the client MAY express its disinterest in the response to such a GET request.
PUT	Suitable for frequent updates (particularly in NON messages) on existing resources. Might not be useful when PUT is used to create a new resource as it may be important for the client to know that the resource creation was actually successful in order to carry out future actions. Also, it may be important to ensure that a resource was actually created rather than updating an existing resource.
POST	If POST is used to update a target resource then No-Response can be used in the same manner as in PUT. This option may also be useful while updating through query strings rather than updating a fixed target resource (see Section 4.1.2.2 for an example).
DELETE	Deletion is usually a permanent action and if the client likes to ensure that the deletion request was properly executed then this option should not be used with the request.

Table 3: Suggested applicability of No-Response for different REST methods

3. Miscellaneous Aspects

This section further describes important implementation aspects worth considering while using the No-Response option. The following discussion contains guidelines and requirements (derived by combining [RFC7252], [RFC7390] and [RFC5405]) for the application developer.

3.1. Re-using Tokens

Tokens provide a matching criteria between a request and the corresponding response. The life of a Token starts when it is assigned to a request and ends when the final matching response is received. Then the Token can again be re-used. However, a request with No-Response typically does not have any guaranteed response path. So, the client has to decide on its own about when it can retire a Token which has been used in an earlier request so that the Token can be reused in a future request. Since the No-Response option is 'elective', a server which has not implemented this option will respond back. This leads to the following two scenarios:

The first scenario is, the client is never going to care about any response coming back or about relating the response to the original request. In that case it MAY reuse the Token value at liberty.

However, as a second scenario, let us consider that the client sends two requests where the first request is with No-Response and the second request, with same Token, is without No-Response. In this case a delayed response to the first one can be interpreted as a response to the second request (client needs a response in the second case) if the time interval between using the same Token is not long enough. This creates a problem in the request-response semantics.

The most ideal solution would be to always use a unique Token for requests with No-Response. But if a client wants to reuse a Token then in most practical cases the client implementation should implement an application specific reuse time after which it can reuse the Token. A minimum reuse time for Tokens with a similar expression as in Section 2.5 of [RFC7390] SHOULD be used:

```
TOKEN_REUSE_TIME = NON_LIFETIME + MAX_SERVER_RESPONSE_DELAY +
                   MAX_LATENCY.
```

NON_LIFETIME and MAX_LATENCY are defined in 4.8.2 of [RFC7252]. MAX_SERVER_RESPONSE_DELAY has same interpretation as in Section 2.5 of [RFC7390] for multicast request. For a unicast request, since the message is sent to only one server, MAX_SERVER_RESPONSE_DELAY means the expected maximum response delay from the particular server to which client sent the request. For multicast requests, MAX_SERVER_RESPONSE_DELAY has the same interpretation as in Section 2.5 of [RFC7390]. So for multicast it is the expected maximum server response delay "over all servers that the client can send a multicast request to". This response delay for a given server includes its specific Leisure period; where Leisure is defined in

Section 8.2 of [RFC7252]. In general, the Leisure for a server may not be known to the client. A lower bound for Leisure, lb_Leisure, is defined in [RFC7252], but not an upper bound as is needed in this case. Therefore the upper bound can be estimated by taking N (N>>1) times the lower bound lb_Leisure:

lb_Leisure = S * G / R

(S = estimated response size; R = data transfer rate; G = group size estimate)

Any estimate of MAX_SERVER_RESPONSE_DELAY MUST be larger than DEFAULT_LEISURE as defined in [RFC7252].

Note: If it is not possible for the client to get a reasonable estimate of the MAX_SERVER_RESPONSE_DELAY then the client, to be safe, SHOULD use a unique Token for each stream of message.

3.2. Taking Care of Congestion

This section provides guidelines for basic congestion control. Better congestion control mechanisms can be designed as future work.

If this option is used with NON messages then the interaction becomes completely open-loop. Absence of any feed-back from the server-end affects congestion-control mechanism. In this case the communication pattern belongs to the class of low-data volume applications as described in Section 3.1.2 of [RFC5405]. More precisely, it maps to the scenario where the application cannot maintain an RTT estimate. Hence, following [RFC5405], a 3 seconds interval is suggested as the minimum interval between successive updates and use even less aggressive rate when possible. However, in case of more frequent update rates the application MUST have some knowledge about the channel and an application developer MUST interweave occasional closed-loop exchanges (e.g. NON messages without No-Response or simply CON messages) to get an RTT estimate between the endpoints.

3.3. Handling No-Response Option for a HTTP-to-CoAP Reverse Proxy

A HTTP-to-CoAP reverse proxy MAY translate an incoming HTTP request to a corresponding CoAP request indicating that no response is required (showing disinterest in all classes of responses) based on some application specific requirement. In this case it is RECOMMENDED that the reverse proxy generates an HTTP response with status code 204 (No Content) when such response is allowed. The

generated response is sent after the proxy has successfully sent out the CoAP request.

In case the reverse proxy applies No-Response for particular class(es) of response(s) it will wait for responses up to an application specific maximum time (T_max) before responding back to the HTTP-side. If a response of a desired class is received within T_max then the response gets translated to HTTP as defined in [I-D.ietf-core-http-mapping]. However if the proxy does not receive any response within T_max, it is RECOMMENDED that the reverse Proxy sends an HTTP response with status code 204 (No Content) when allowed for the specific HTTP request method.

4. Exemplary Application Scenarios

This section describes some exemplary application scenarios which may potentially benefit from the use of No-Response option.

4.1. Frequent Update of Geo-location from Vehicles to Backend Server

Let us consider an intelligent traffic system (ITS) consisting of vehicles equipped with a sensor-gateway comprising sensors like GPS and Accelerometer. The sensor-gateway acts as a CoAP client end-point. It connects to the Internet using a low-bandwidth cellular (e.g. GPRS) connection. The GPS co-ordinates of the vehicle are periodically updated to the backend server.

While performing frequent location update, retransmitting (through the CoAP CON mechanism) a location co-ordinate which the vehicle has already left in the meantime is not efficient as it adds redundant traffic to the network. Therefore, the updates are done using NON messages. However, given the huge number of vehicles updating frequently, the NON exchange will also trigger huge number of responses from the backend. Thus the cumulative load on the network will be quite significant.

On the contrary, if the client end-points on the vehicles explicitly declare that they do not need any status response back from the server then load will be reduced significantly. The assumption is that, since the update rate is high, stray losses in geo-location reports will be compensated with the large update rate.

Note: It may be argued that the above example application can also be implemented using Observe option ([RFC7641]) with NON notifications. But, in practice, implementing with Observe would require lot of book-keeping at the data-collection end-point at the backend (observer). The observer needs to maintain all the

observe relationships with each vehicle. The data collection end-point may be unable to know all its data sources beforehand. The client end-points at vehicles may go offline or come back online randomly. In case of Observe the onus is always on the data collection end-point to establish an observe relationship with each data-source. On the other hand, implementation will be much simpler if the initiative is left on the data-source to carry out updates using No-Response option. Putting it another way: the implementation choice depends on the perspective of interest to initiate the update. In an Observe scenario the interest is expressed by the data-consumer. On the contrary, the classic update case applies when it is the interest of the data-producer. The 'No-Response' option enables to make classic updates further less resource consuming.

Following subsections illustrate some exemplary exchanges based on the application described above.

4.1.1. Using No-Response with PUT

Each vehicle is assigned a dedicated resource: vehicle-stat-<n>, where <n> can be any string uniquely identifying the vehicle. The update requests are sent over NON type of messages. The No-Response option causes the server not to respond back.

```
Client Server
|      |
|      |
+----->| Header: PUT (T=NON, Code=0.03, MID=0x7d38)
| PUT  | Token: 0x53
|      | Uri-Path: "vehicle-stat-00"
|      | Content Type: text/plain
|      | No-Response: 127
|      | Payload:
|      | "VehID=00&RouteID=DN47&Lat=22.5658745&Long=88.4107966667&
|      | Time=2013-01-13T11:24:31"
|      |
[No response from the server. Next update in 20 secs.]
|      |
+----->| Header: PUT (T=NON, Code=0.03, MID=0x7d39)
| PUT  | Token: 0x54
|      | Uri-Path: "vehicle-stat-00"
|      | Content Type: text/plain
|      | No-Response: 127
|      | Payload:
|      | "VehID=00&RouteID=DN47&Lat=22.5649015&Long=88.4103511667&
|      | Time=2013-01-13T11:24:51"
```

Figure 1: Exemplary unreliable update with No-Response option using PUT.

4.1.2. Using No-Response with POST

4.1.2.1. POST updating a fixed target resource

In this case POST acts the same way as PUT. The exchanges are same as above. The updated values are carried as payload of POST as shown in Figure 2.

```
Client Server
|      |
|      |
+----->| Header: POST (T=NON, Code=0.02, MID=0x7d38)
| POST | Token: 0x53
|      | Uri-Path: "vehicle-stat-00"
|      | Content Type: text/plain
|      | No-Response: 127
|      | Payload:
|      | "VehID=00&RouteID=DN47&Lat=22.5658745&Long=88.4107966667&
|      | Time=2013-01-13T11:24:31"
|      |
[No response from the server. Next update in 20 secs.]
|      |
+----->| Header: PUT (T=NON, Code=0.02, MID=0x7d39)
| POST | Token: 0x54
|      | Uri-Path: "vehicle-stat-00"
|      | Content Type: text/plain
|      | No-Response: 127
|      | Payload:
|      | "VehID=00&RouteID=DN47&Lat=22.5649015&Long=88.4103511667&
|      | Time=2013-01-13T11:24:51"
```

Figure 2: Exemplary unreliable update with No-Response option using POST as the update-method.

4.1.2.2. POST updating through query-string

It may be possible that the backend infrastructure deploys a dedicated database to store the location updates. In such a case the client can update through a POST by sending a query string in the URI. The query-string contains the name/value pairs for each update. 'No-Response' can be used in same manner as for updating fixed resources. The scenario is depicted in Figure 3.

```
Client Server
|      |
|      |
+----->| Header: POST (T=NON, Code=0.02, MID=0x7d38)
| POST | Token: 0x53
|      | Uri-Path: "updateOrInsertInfo"
|      | Uri-Query: "VehID=00"
|      | Uri-Query: "RouteID=DN47"
|      | Uri-Query: "Lat=22.5658745"
|      | Uri-Query: "Long=88.4107966667"
|      | Uri-Query: "Time=2013-01-13T11:24:31"
|      | No-Response: 127
|      |
[No response from the server. Next update in 20 secs.]
|      |
+----->| Header: POST (T=NON, Code=0.02, MID=0x7d39)
| POST | Token: 0x54
|      | Uri-Path: "updateOrInsertInfo"
|      | Uri-Query: "VehID=00"
|      | Uri-Query: "RouteID=DN47"
|      | Uri-Query: "Lat=22.5649015"
|      | Uri-Query: "Long=88.4103511667"
|      | Uri-Query: "Time=2013-01-13T11:24:51"
|      | No-Response: 127
|      |
```

Figure 3: Exemplary unreliable update with No-Response option using POST with a query-string to insert update information to backend database.

4.2. Multicasting Actuation Command from a Handheld Device to a Group of Appliances

A handheld device (e.g. a smart phone) may be programmed to act as an IP enabled switch to remotely operate on a single or group of IP enabled appliances. For example, send a multicast request to switch on/ off all the lights of a building. In this case the IP switch application can use the No-Response option in a NON request message to reduce the traffic generated due to simultaneous CoAP responses from all the lights.

Thus No-Response helps in reducing overall communication cost and the probability of network congestion in this case.

4.2.1. Using Granular Response Suppression

The IP switch application may optionally use granular response suppression such that the error responses are not suppressed. In that case the lights which could not execute the request would respond back and be readily identified. Thus, explicit suppression of option classes by the multicast client may be useful to debug the network and the application.

5. IANA Considerations

The IANA has assigned number 284 to this option in the CoAP Option Numbers registry:

Number	Name	Reference
284	No-Response	Section 2 of this document

6. Security Considerations

The No-Response option defined in this document presents no security considerations beyond those in Section 11 of the base CoAP specification [RFC7252].

7. Acknowledgments

Thanks to Carsten Bormann, Matthias Kovatsch, Esko Dijk, Bert Greevenbosch, Akbar Rahman and Klaus Hartke for their valuable inputs.

8. References

8.1. Normative References

[RFC7252]

Shelby, Z., Hartke, K. and Bormann, C.,"Constrained Application Protocol (CoAP)", RFC 7252, June, 2014

[RFC7641]

Hartke, K.," Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, September, 2015

[RFC7390]

Rahman, A. and Dijk, E.,"Group Communication for CoAP", RFC 7390, October, 2014

[RFC5405]

Eggert, L. and Fairhurst, G.," Unicast UDP Usage Guidelines for Application Designers", RFC 5405, November, 2008

[I-D.ietf-core-http-mapping]

Castellani, A., et al., "Guidelines for HTTP-CoAP Mapping Implementations", draft-ietf-core-http-mapping-07, July 3, 2015

8.2. Informative References

[MOBIQUITOUS 2013]

Bhattacharyya, A., Bandyopadhyay, S. and Pal, A., "ITS-light: Adaptive lightweight scheme to resource optimize intelligent transportation tracking system (ITS)-Customizing CoAP for opportunistic optimization", 10th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (Mobiquitous 2013), December, 2013.

[Sensys 2013]

Bandyopadhyay, S., Bhattacharyya, A. and Pal, A., "Adapting protocol characteristics of CoAP using sensed indication for vehicular analytics", 11th ACM Conference on Embedded Networked Sensor Systems (Sensys 2013), November, 2013.

Authors' Addresses

Abhijan Bhattacharyya
Tata Consultancy Services Ltd.
Kolkata, India

Email: abhijan.bhattacharyya@tcs.com

Soma Bandyopadhyay
Tata Consultancy Services Ltd.
Kolkata, India

Email: soma.bandyopadhyay@tcs.com

Arpan Pal
Tata Consultancy Services Ltd.
Kolkata, India

Email: arpan.pal@tcs.com

Tulika Bose
Tata Consultancy Services Ltd.
Kolkata, India

Email: tulika.bose@tcs.com