CoRE
Internet Draft
Intended status: Standards track
Expires: February 2015

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

August 5, 2014

CoAP option for no server-response
draft-tcs-coap-no-response-option-07

Abstract

There can be typical M2M scenarios where responses from the data sink to the data source against request from the source might be considered redundant. This kind of open-loop exchange (with no reverse path from the sink to the source) may be desired while updating resources in constrained systems looking for maximized throughput with minimized resource consumption. CoAP already provides a non-confirmable (NON) mode of exchange where the receiving end-point does not respond with ACK. However, the receiving end-point responds the sender with a status code indicating "the result of the attempt to understand and satisfy the request".

This draft introduces a header option: 'No-Response' to suppress responses from the receiver and discusses exemplary use cases which motivated this proposition based on real experience. This option also provides granularity by allowing suppression of a typical class or a combination of classes of responses. This option may be effective for both unicast and multicast scenarios.

Status of this Memo

Table of Contents

1. Introduction

This draft proposes a new header option 'No-Response' for Constrained Application Protocol (CoAP) [RFC7252]. This option enables the sender end-point to explicitly express its disinterest in getting responses back from the receiving end-point. By default this option expresses disinterest in any kind of response. This option should be applicable along with non-confirmable (NON) updates. At present this option will have no effect if used with confirmable (CON) mode.

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

1.1. Granular suppression of responses

This option enables granularity by allowing the sender to choose the typical class or combination of classes of responses which it is disinterested in. For example, a sender 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 sender. No response is required in case of 2.xx classes. A similar scheme is described in Section 3.7 of [I-D.ietf-core-groupcomm] on the server side. Here the server may perform granular suppression for group communication. But in that case the server itself decides whether to suppress responses or not. This option enables the clients to explicitly inform the server about the disinterest in responses.

1.2. Terminology

The terms used in this draft 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. Potential benefits

If this option is opportunistically used with fitting M2M applications then the concerned systems may benefit in the following aspects:

* Reduction in network clogging by effectively reducing the overall traffic.

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

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

* Reduction in communication cost.

* Help satisfy hard real-time requirements since waiting due to closed loop latency MAY be completely avoided.

3. Exemplary application scenarios

Next sub-sections describe some exemplary user stories which may potentially benefit by using No-Response option.

3.1. Frequent update of geo-location from vehicles to backend

Let us consider an intelligent traffic system (ITS) consisting of vehicles each of which is equipped with a sensor-gateway comprising sensors like GPS and Accelerometer. The sensor-gateway connects to the Internet using a low-bandwidth cellular (e.g. GPRS) connection. The GPS co-ordinates are periodically updated to the backend server by the gateway. The update rate in case of ITS is adaptive to the motional-state of the vehicle. If the vehicle moves fast the update rate is high as the position of the vehicle changes rapidly. If the vehicle is static or moves slowly then the update rate is low. This ensures that bandwidth and energy is not consumed unnecessarily. The motional-state of the vehicle is inferred by a local analytics running on the sensor-gateway which uses the accelerometer data and the rate of change in GPS co-ordinates. The back-end server hosts applications which use the updates for each vehicle and produce necessary information for remote users.

Retransmitting a location co-ordinate which is already passed by a vehicle is not efficient as it adds redundant traffic to the network. So, the updates are done in NON mode. However, given the thousands of vehicles updating frequently, the NON exchange will also trigger huge number of status responses from the backend. Each response in the air is of 4 bytes of application layer plus several bytes originating from the lower layers. Thus the cumulative load on the network will be quite significant.

On the contrary, if the edge devices explicitly declare that they do not need any status response then significant load will be reduced from the network and the server as well. The assumption is that since the update rate is high stray losses in geo-locations will be compensated with the large update rate and thereby not affecting the end applications.

Mapping the above scenario to the benefits mentioned in Section 2 reveals that use of 'No-Response' will help in:

* Reduction in network clogging

* Reduction in server-side loading

* Help in achieving real-time requirements as the application is not bound by any delay due to closed loop latency

3.2. 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 the smart phone can be programmed to send a multicast request to switch on/ off all the lights of a building. In this case the IP switch application can uses No-Response option along with NON to reduce the traffic generated due to simultaneous status responses from hundreds of lights.

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

3.2.1. 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.

4. Option Definition

The properties of this option are as in Table 1.

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

Table 1: Option Properties

This option is Elective and Non-Repeatable. This is a request option and primarily intended to be used with non-confirmable update requests (e.g., PUT) and should have no effect if used with a CON request. This option is not applicable and should have no effect for usual GET requests asking for resource representation. However, this option MAY be used with special GET request for 'cancellation' of an observe session (Section 3.6 of [I-D.ietf-core-observe]). This option contains values to optionally indicate interest/ disinterest in all or a particular class or combination of classes of responses as described in the next sub-section.

The following table provides a 'ready-reference' on possible applicability of this option for all the four REST methods. This table is prepared in view of the type of application scenarios foreseen so far.

Method Name	Remarks on applicability
GET	This option does not apply to GET under usual circumstances when the client requests the contents of a resource. However, this option MAY be useful for special GET requests. At present only one such application is identified which is the 'cancellation' procedure for 'Observe'. Observe-cancellation requires a client to issue a GET request which has the same token as the token of the original observe request and includes an Observe Option with the value set to 'deregister' (1). In this case the server response does not contain any payload. Under such situation the client MAY express its disinterest in the response from the server.
PUT	Mostly suitable for frequent updates in NON mode on existing resources. Might not be useful when PUT creates a new resource.
POST	If POST is used just to update a target resource then No-Response can be used in the same manner as in NON-PUT. May also be applicable when POST performs resource creation and the client does not refer to the resource in future. For example, than updating a fixed resource, POST API may contain a query-string with name/value pairs for a defined action (ex. insertion into a database as part of frequent updates). The resources created this way may be 'short-lived' resources which the client will not refer to in future (see Section 6.1.2.2).
DELETE	Deletion is usually a permanent action and the client SHOULD make sure that the deletion actually happened. SHOULD NOT be applicable.

Table 2: Applicability of No-Response for different methods

4.1. Granularity in response suppression

This option is defined as a bit-map (Table 3) to achieve granular suppression.

Value	Binary Representation	Description
0	00000000	Suppress all responses (same as empty value).
2	00000010	Allow 2.xx success responses.
8	00001000	Allow 4.xx client errors.
16	00010000	Allow 5.xx server errors.

Table 3: Option values

XOR of the values defined for allowing particular classes will result in allowing a combination of classes of responses. So, a value of 18 (binary: 00010010) will result in allowing all 2.xx and 5.xx classes of responses. It is to be noted that a value of 26 will indicate that all types of responses are to be allowed (which is as good as not using No-Response at all).

Implementation Note: The use of No-Response option is very much driven by the application scenario and the characteristics of the information to be updated. Judicious use of this option benefits the overall system as explained in Sections 2 and 3.

When No-Response is used with empty or 0 value, the updating end-point should cease the listening activity for response against the particular request. On the contrary, opening up at least one class of responses means that the updating end-point can no longer stop listening and must be configured to listen up to some application specific time-out period for the particular request. The updating end-point never knows whether the present update will be a success or a failure. Thus, if the client decides to open up the responses for errors (4.xx & 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). This kind of situation may arise for the scenario in Section 3.2.1. Under such circumstances the use of No-Response may not help improving the performance in terms of overall latency. However, the advantages in terms of saving network and energy resources will still hold.

A point to be noted in view of the above example is that there may be situations when the response on errors might get lost. In such a situation the sender would wait up to the time-out period

but will not receive any response. But this should not lead to the impression to the sender that the request was successful. The situation will worsen if the receiver is no longer active. The application designer needs to tackle such situation. For example, the sender may strategically insert requests without No-Response after N numbers of requests with No-Response.

5. Miscellaneous aspects

This section further describes few important implementation aspects worth considering while using No-Response. The following discussion does not mandate anything, rather provides suggestive guidelines for the application developer.

5.1. Re-use interval for message IDs

Since No-Response is primarily based on CoAP-NON, 'NON-LIFETIME' (as defined in Section 4.8.2 of [RFC7252]) is suggested as the time interval over which a message ID can be safely re-used.

5.2. 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 NON request with No-Response does not have any 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 MAY emanate a response. This leads to the following two scenarios:

The first scenario is when 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 gap between using the same tokens is not 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 'patience' time till which it can re-use the token. Appendix-B.4.1 of [I-D.draft-bormann-coap-misc] refers to the 'patience' option defined in [I-D.draft-li-coap-patience]. 'Patience' option effectively puts a deadline to the server to respond back. However, 'patience' is not exposed to the protocol level at present. This draft suggests a reuse time for tokens with similar expression as in Section 2.5 of [I-D.ietf-core-groupcomm]:

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 [I-D.ietf-core-groupcomm] for multicast request. But for unicast request MAX_SERVER_RESPONSE_DELAY is simply the expected maximum response delay from the server to which client sent the request. This delay includes the maximum Leisure time period as defined in Section 8.2 of [RFC7252] and Appendix-B.4.2 of [I-D.draft-bormann-coap-misc]where group size (G) = 1 for unicast request.

If it is not possible for the client to get a reasonable estimate of the MAX_SERVER_RESPONSE_DELAY then the client SHOULD use a unique token for the request with No-Response to be safe.

5.3. Taking care of congestion

The possible communication scenarios leveraging the benefits of 'No-Response' should primarily fall into the class of low-data volume applications as described in Section 3.1.2 of [RFC5405]. Precisely, they should map to the scenario where the application cannot maintain an RTT estimate. Hence, following [RFC5405], a 3s interval is suggested as the minimum interval between successive updates. However, an application developer MAY interweave occasional closed-loop exchanges (e.g. CoAP-NON without No-Response or CoAP-CON) to get an RTT estimate between the end-points and adjust time-to-time the interval between updates.

5.4. Duality with the 'Observe' option

Unlike the multicast actuation scenarios (Section 3.2), scenarios like frequent update using No-Response leads to an interesting observation. The 'No-Response' option in a sense complements the 'Observe' option with NON-notifications ([I-D.ietf-core-observe]).

In case of the later the update notifications from the server reach the observer client without triggering any response from the observer. However, there is a difference in the point of interest. In the 'Observe' scenario the interest is expressed by the 'consumer' to get the data. On the contrary, the updates using 'No-Response' applies to the scenario when it is the interest of the 'producer' to update the data. It is up to the application designer to choose between No-Response and 'observe' with notifications in NON mode. For example, the scenario of location update described in Section 3.1 above might also be deployed using observe with NON-notifications. In that case the backend infrastructure would have to subscribe to each individual sensor gateway at the vehicles. But, the 'book-keeping' exercise required at the backend for such an implementation may not be very trivial and deployment with No-Response may be far more straight-forward. However, 'No-Response' and 'Observe' using NON-notification may be combined together, under permitting condition, to achieve high performance gain in an end-to-end producer-consumer application. A typical example is illustrated in Section 6.2.

6. Example

This section illustrates few examples of exchanges based on the scenario narrated in Section 3.1. Examples for other scenarios can be easily conceived based on these illustrations.

6.1. Request/response Scenario

6.1.1. Using No-Response with PUT

Figure 1 shows a typical request with this option. The depicted scenario occurs when the vehicle#n moves very fast and update rate is high. The vehicle is assigned a dedicated resource: vehicle-stat-<n>, where <n> can be any string uniquely identifying the vehicle. The update requests are in NON mode. 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: 0
   |      | 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: 0
   |      | 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.

6.1.2. Using No-Response with POST

POST "usually results in a new resource being created or the target resource being updated". Exemplary uses of 'No-Response' for both these usual actions of POST are given below.

6.1.2.1. POST updating a 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: 0
   |       | 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: 0
   |       | 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.

6.1.2.2. POST performing updates through resource creation

In most practical implementations the backend infrastructure as described in Section 3.1 will have a dedicated database to store the location updates. In such a case the client would send an update string as the POST URI which contains the name/value pairs for each update. Thus frequent updates may be performed through POST by creating such 'short-lived' resources which the client would not refer to in future. Hence '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: "insertInfo"
   |       | 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: 0
   |       |
   [No response from the server. Next update in 20 secs.]
   |       |
   +------>| Header: POST (T=NON, Code=0.02, MID=0x7d39)
   | POST  | Token: 0x54
   |       | Uri-Path: "insertInfo"
   |       | 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: 0
   |       |
```

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

6.2. An end-to-end system combining No-Response and Observe

This example illustrates the scenario pointed out in Section 5.3 above. The 'No-Response' option can be combined with the 'Observe' option with NON-notifications to create a lightweight end-to-end producer-consumer system. For example, the vehicular updates from a remote vehicle may be observed by a remote observer in a PDA as shown in figure 4.

```
   Producer Server                                         Consumer
   (Client)                                                (Client)
   |        |                                                    |
   |        |                                               <-----+
   |        |                                                GET  |
   +----->  |                          (Observe: empty, Token: 30)|
   | POST   |                                                    |
   |        | Header: POST (T=NON, Code=0.02, MID=0x7d38)         |
   |        | Token: 0x53                                         |
   |        | Uri-Path: "insertInfo"                              |
   |        | 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: 0                                      |
   |        |                                                    |
   |        +----->                                              |
   |        | 2.05 (T=NON, MID=0x5d40, Token: 30)                 |
   |        |     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: POST (T=NON, Code=0.02, MID=0x7d39)         |
   | POST   | Token: 0x54                                         |
   |        | Uri-Path: "insertInfo"                              |
   |        | 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: 0                                      |
   |        |                                                    |
   |        +----->                                              |
   |        | 2.05 (T=NON, MID=0x5d41, Token: 30)                 |
   |        |     Payload:                                       |
   |        |     "VehID=00&RouteID=DN47&Lat=22.5649015&         |
   |        |      Long=88.4103511667& Time=2013-01-13T11:24:51"|
   |        |                                                    |
```

Figure 4: Exemplary end-to-end update and observe scenario using 'No-Response' for NON-updates from 'producer' and observe with NON-notifications by the 'consumer'.

7. IANA Considerations

The IANA is requested to add the following option number entries:

Number	Name	Reference
92	No-Response	Section 4 of this document

8. 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].

9. Acknowledgments

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

10. References

10.1. Normative References

[RFC7252]

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

[I-D.ietf-core-observe]

Hartke, K.,"Observing Resources in CoAP", draft-ietf-core-observe-14, June 30, 2014

[I-D.ietf-core-groupcomm]

Rahman, A. and Dijk, E.,"Group Communication for CoAP", draft-ietf core-groupcomm-21, July 31, 2014

[I-D.draft-bormann-coap-misc]

Bormann, C. and Hartke, K., "Miscelleneous additions to CoAP", draft-bormann-coap-misc-26, December 19, 2013

[I-D.draft-kovatsch-lwig-coap]

Kovatsch, M., Bergmann, O., Dijk, E., He, X. and Bormann, C., "CoAP Implementation Guidance", draft-kovatsch-lwig-coap-03, February 28, 2014

[RFC5405]

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

[I-D.draft-li-coap-patience]

Li, K., Greevenbosch, B., Dijk, E. and Loreto, S.," CoAP Option Extension: Patience", draft-li-core-coap-patience-option-04, July 04, 2014

10.2. 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