core
Internet-Draft
Intended status: Informational
Expires: September 16, 2016

P. van der Stok
A. Sehgal
Consultant

March 15, 2016

Patch Method for Constrained Application Protocol (CoAP)
draft-vanderstok-core-patch-03

Abstract

The existing Constrained Application Protocol (CoAP) PUT method only allows a complete replacement of a resource. This does not permit applications to perform partial resource modifications. In case of resources with larger or complex data, or in situations where a resource continuity is required, replacing a resource is not an option. Several applications using CoAP will need to perform partial resource modifications. This proposal adds new CoAP methods, PATCH and iPATCH, to modify an existing CoAP resource partially.

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 September 16, 2016.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
1.2. Terminology and Acronyms 3
2. PATCH (iPATCH) Method 3
2.1. A Simple PATCH (iPATCH) Example 4
2.2. Response Codes 7
2.3. Option Numbers 7
3. Error Handling 8
4. Security Considerations 9
5. IANA Considerations 10
6. Acknowledgements 10
7. Change log 10
8. References 11
8.1. Normative References 11
8.2. Informative References 12
Authors' Addresses 12

1. Introduction

This specification defines the new Constrained Application Protocol (CoAP) [RFC7252] methods, PATCH and iPATCH, which are used to apply partial modifications to a resource.

PATCH is also specified for HTTP in [RFC5789]. Most of the motivation for PATCH described in [RFC5789] also applies here. iPATCH is the idem-potent version of PATCH.

The PUT method exists to overwrite a resource with completely new contents, and cannot be used to perform partial changes. When using PUT for partial changes, proxies and caches, and even clients and servers, may get confused as to the result of the operation. PATCH was not adopted in an early design stage of CoAP, however, it has become necessary with the arrival of applications that require partial updates to resources (e.g. [I-D.vanderstok-core-comi]). Using PATCH avoids transferring all data associated with a resource in case of modifications, thereby not burdening the constrained communication medium.

This document relies on knowledge of the PATCH specification for HTTP [RFC5789]. This document provides extracts from [RFC5789] to make independent reading possible.

1.1. Requirements Language

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].

1.2. Terminology and Acronyms

This document uses terminology defined in [RFC5789] and [RFC7252].

2. PATCH (iPATCH) Method

The PATCH (iPATCH) method requests that a set of changes described in the request payload is applied to the target resource of the request. The set of changes is represented in a format identified by a media type. If the Request-URI does not point to an existing resource, the server MAY create a new resource with that URI, depending on the patch document type (whether it can logically modify a null resource) and permissions, etc. Creation of a new resource would result in a 2.01 (Created) Response Code dependent of the patch document type.

Restrictions to a PATCH (iPATCH) can be made by including the If-Match or If-None-Match options in the request (see Section 5.10.8.1 and 5.10.8.2 of [RFC7252]). If the resource could not be created or modified, then an appropriate Error Response Code SHOULD be sent.

The difference between the PUT and PATCH requests is extensively documented in [RFC5789].

PATCH is not safe and not idempotent conformant to HTTP PATCH specified in [RFC5789].

iPATCH is not safe but idempotent conformant to CoAP PUT specified in [RFC7252], Section 5.8.3.

An iPATCH request is idempotent to prevent bad outcomes from collisions between two iPATCH requests on the same resource in a similar time frame. These collisions can be detected with the MessageId and the source end-point provided by the CoAP protocol (see section 4.5 of [RFC7252].

PATCH and iPATCH are both atomic. The server MUST apply the entire set of changes atomically and never provide a partially modified representation to a concurrently executed GET request. Given the constrained nature of the servers, most servers will only execute CoAP requests consecutively, thus preventing a concurrent partial overlapping of request modifications. Resuming, modifications MUST NOT be applied to the server state when an error occurs or only a

partial execution is possible on the resources present in the server. When the PATCH request is over-specified (i.e. Request specifies modifications to attributes which do not exist in the server), The server MAY execute all modifications to existing attributes and return a response code 2.02 Accepted.

The atomicity applies to a single server. When a PATCH (iPATCH) request is multicast to a set of servers, each server can either execute all required modifications or not. It is not required that all servers execute all modifications or none. An Atomic Commit protocol that provides multiple server atomicity, is out of scope.

A PATCH (iPATCH) response can invalidate a cache conformant with the PUT response. Caching behaviour as function of the valid 2.xx response codes for PATCH (iPATCH) are:

A 2.01 (Created) response invalidates any cache entry for the resource indicated by the Location-* Options; the payload is a representation of the action result.

A 2.04 (Changed) response invalidates any cache entry for the target resource; the payload is a representation of the action result.

There is no guarantee that a resource can be modified with PATCH (iPATCH). Servers are required to support a subset of the content formats as specified in sections 12.3 and 5.10.3 of [RFC7252]. Servers MUST ensure that a received PATCH payload is appropriate for the type of resource identified by the target resource of the request.

Clients MUST choose to use PATCH (iPATCH) rather than PUT when the request affects partial updates of a given resource.

PATCH (iPATCH) MUST not be used to restore default values to resource attributes which are not specified in the payload. PATCH (iPATCH) specifically guarantees that unspecified resource attributes are not changed.

2.1. A Simple PATCH (iPATCH) Example

The example is taken over from [RFC6902], which specifies a JSON notation for PATCH operations. A resource located at www.example.com/object contains a target JSON document.

```
JSON document original state
            {
              "x-coord": 256,
              "y-coord": 45",
              "foo": ["bar","baz"]
            }

 REQ:
     iPATCH CoAP://www.example.com/object
Content-Format: application/json-patch+json
          [
              { "op":"replace","path":"x-coord","value":45}
          ]
RET:
     CoAP 2.04 Changed

JSON document final state
            {
               "x-coord": 45,
               "y-coord": 45,
               "foo": ["bar","baz"]
             }
```

This example illustrates use of an idempotent modification to the x-coord attribute of the existing resource "object". The 2.04 (Changed) response code is conform with the CoAP PUT method.

The same example using the Content-Format application/merge-patch+json from [RFC7396] looks like:

```
JSON document original state
            {
              "x-coord": 256,
              "y-coord": 45,
              "foo": ["bar","baz"]
            }

 REQ:
     iPATCH CoAP://www.example.com/object
Content-Format: application/merge-patch+json
     { "x-coord":45}
RET:
     CoAP 2.04 Changed

JSON document final state
            {
               "x-coord": 45,
               "y-coord": 45,
               "foo": ["bar","baz"]
             }
```

The examples show the use of the iPATCH method, but the use of the PATCH method must have led to the same result. Below a non-idempotent modification is shown. Because the action is non-idempotent, iPATCH returns an error, while PATCH executes the action.

```
   JSON document original state
               {
                 "x-coord": 256,
                 "y-coord": 45",
                 "foo": ["bar","baz"]
               }

    REQ:
        iPATCH CoAP://www.example.com/object
   Content-Format: application/json-patch+json
            [
                { "op":"add","path":"foo/1","value":"bar"}
            ]
   RET:
        CoAP 4.12 Precondition Failed

   JSON document final state is unchanged

   REQ:
        PATCH CoAP://www.example.com/object
   Content-Format: application/json-patch+json
            [
                { "op":"add","path":"foo/1","value":"bar"}
            ]
   RET:
        CoAP 2.04 Changed

   JSON document final state

               {
                  "x-coord": 45,
                  "y-coord": 45,
                  "foo": ["bar","bar","baz"]
                }
```

2.2. Response Codes

PATCH (iPATCH) for CoAP adopt the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252].

2.3. Option Numbers

PATCH for CoAP adopts the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

3. Error Handling

A PATCH (iPATCH) request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH (iPATCH) payload: If a server determines that the payload provided with a PATCH (iPATCH) request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported PATCH (iPATCH) payload: In case a client sends payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a PATCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 4.22 (Unprocessable Entity) CoAP error. More specific scenarios might include situations when:

* the server has insufficient computing resources to complete the request successfully -- 4.13 (Request Entity Too Large) CoAP Response Code,

* the resource specified in the request becomes invalid by applying the payload -- 4.06 (Not Acceptable) CoAP Response Code,

In case there are more specific errors that provide more insight into the problem, then those should be used.

Resource not found: The 4.04 (Not Found) error should be returned in case the payload of a PATCH request cannot be applied to a non-existent resource.

Failed precondition: In case the client uses the conditional If-Match or If-None-Match option to define a precondition for the PATCH request, and that precondition fails, then the server can return the 4.12 (Precondition Failed) CoAP error.

Request too large: If the payload of the PATCH request is larger than a CoAP server can process, then it can return the 4.13 (Request Entity Too Large) CoAP error.

Conflicting state: If the modification specified by a PATCH (iPATCH) request causes the resource to enter an inconsistent state that the server cannot resolve, the server can return the 4.09 (Conflict) CoAP response. The server SHOULD generate a payload that includes enough information for a user to recognize the source of the conflict. The server MAY return the actual resource state to provide the client with the means to create a new consistent resource state. Such a situation might be encountered when a structural modification is applied to a configuration data-store, but the structures being modified do not exist.

Concurrent modification: Resource constrained devices might need to process requests in the order they are received. In case requests are received concurrently to modify the same resource but they cannot be queued, the server can return a 5.03 (Service unavailable) CoAP response code.

Conflict handling failure: If the modification implies the reservation of resources or the waiting on conditions to become true, leading to a too long request execution time, the server can return 5.03 (service unavailable) response code.

It is possible that other error situations, not mentioned here, are encountered by a CoAP server while processing the PATCH request. In these situations other appropriate CoAP status codes can also be returned.

4. Security Considerations

This section analyses the possible threats to the CoAP PATCH (iPATCH) protocol. It is meant to inform protocol and application developers about the security limitations of CoAP PATCH (iPATCH) as described in this document. The security consideration of section 15 of [RFC2616], section 11 of [RFC7252], and section 5 of [RFC5789] also apply.

The security considerations for PATCH (iPATCH) are nearly identical to the security considerations for PUT ([RFC7252]). The mechanisms used for PUT can be used for PATCH (iPATCH) as well.

PATCH (iPATCH) is secured following the CoAP recommendations as specified in section 9 of [RFC7252]. When more appropriate security techniques are standardized for CoAP, PATCH (iPATCH) can also be secured by those new techniques.

5. IANA Considerations

The entry with name PATCH in the sub-registry, "CoAP Method Codes", is 0.05. The entry with name iPATCH in the sub-registry, "CoAP Method Codes", is 0.06. The additions will follow the "IETF Review or IESG Approval" procedure as described in [RFC5226].

A new response code must be entered to the sub-registry "CoAP response codes" which apply to the methods PATCH and iPATCH.

Code 4.09 with Description "Conflict" and described in this specification.

The addition to this sub-registry will follow the "IETF Review or IESG Approval" as described in [RFC5226].

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for the following media type formats: "application/json-patch+json" [RFC6902], and "application/merge-patch+json" [RFC7396].

6. Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP. We are grateful for discussions with Christian Amsuss, Carsten Bormann, Timothy Carey, Paul Duffy, Kovatsch Matthias, Michel Veillette, Michael Verschoor, Thomas Watteyne, and Gengyu Wei.

7. Change log

When published as a RFC, this section needs to be removed.

Version 0 to version 1:

- Changed patch motivation text.
- Removed sub-resource concept.
- Updated cache handling.
- Extended example.
- Update of error handling.

Version 1 to version 2

- section 3 rephrased use of error 4.09

- added conflict handling failure
- added idempotent iPATCH method

Version 2 to version 3

- added line about default values
- content-Type to content_Format
- extended Patch example with non-idempotent request

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <http://www.rfc-editor.org/info/rfc5789>.

[RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013, <http://www.rfc-editor.org/info/rfc6902>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <http://www.rfc-editor.org/info/rfc7396>.

8.2. Informative References

[I-D.vanderstok-core-comi]
Stok, P. and A. Bierman, "CoAP Management Interface", draft-vanderstok-core-comi-09 (work in progress), March 2016.

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

Anuj Sehgal
Consultant

Email: anuj@iurs.org