core
Internet-Draft
Intended status: Standards Track
Expires: December 19, 2013

I. Ishaq
J. Hoebeke
F. Van den Abeele
iMinds-IBCN/UGent

June 17, 2013

CoAP Entities
draft-ishaq-core-entities-00

Abstract

This document describes a format to create entities that can be used for group communication using CoAP unicast messages.

Note

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

Status of This Memo

Copyright Notice

Table of Contents

1. Requirements notation

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].

The above key words are used to establish a set of guidelines for CoAP entities. An implementation of CoAP entities MAY implement these guidelines; an implementation claiming compliance to this document MUST implement the set of guidelines.

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap] and [I-D.greevenbosch-core-profile-description]. In addition, this document defines the following terminology:

Entity
  A group of resources on CoAP servers that can be created, used or manipulated through a single CoAP request.

Entity Manager (EM)

The component that manages the entities. This component, which can reside e.g. on the Border Gateway of the LLN, is responsible for maintaining entities. Clients on the Internet can interact with an EM to create new entities and/or customize how these entities should behave.

2. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a RESTful protocol for constrained nodes. The networks that connect these nodes together are often referred to as low power and lossy networks (LLNs).

Typically, each of the constrained servers has at least one CoAP resource that may be queried by clients to obtain information about the nodes themselves (e.g. battery level), about the environment that they monitor (e.g. temperature of the room), or to trigger the nodes to perform real-world actions (switch the light on).

Depending on the application, information from individual nodes might not be sufficient, reliable, or useful. An application may need to aggregate and/or compare data from several nodes in order to obtain accurate results. In the same way, a single user request might need to trigger a series of actions on multiple actuators to perform a single user request.

Although multicast may be used to transmit the same request to several nodes [I-D.ietf-core-groupcomm], multicast communication in LLNs has some disadvantages. For instance, it is difficult to avoid duplication of messages, and duplication is undesirable in an LLN where bandwidth is limited for these constrained nodes. Furthermore, basic multicast is not reliable in an LLN, which is problematic for requests that require guaranteed delivery. Security of multicast is another issue. Currently CoAP relies on Datagram Transport Layer Security (DTLS) [RFC6347] for secure unicast communication. At the moment, DTLS requires non-standard extensions like [I-D.keoh-tls-multicast-security] to secure multicast. As demonstrated by the formation of the upcoming DTLS-IoT WG (pending a BoF at IETF 87) that aims to introduce multicast record layer support for DTLS, work is very much ongoing in this field but no standard solution is available as of today. Also, the creation of multicast groups, defining which nodes should be addressed when using a particular multicast address, is hard to realize inside LLNs. For instance, the approach in [I-D.ietf-core-groupcomm] suggests that every CoAP endpoint should implement the "core.gp" interface. Additionally, the use of multicast increases the footprint of the code that needs to fit on the constrained nodes, and it is to be expected that this functionality will not be available in many LLNs.

Consequently, in some cases the use of multicast might be not feasible or provide a suboptimal solution.

As an alternative, unicast-based solutions should be considered. Simple unicast solutions are defined in the CoRE Interfaces draft [I-D.ietf-core-interfaces]. Among other interface types, this draft defines the Batch interface type and its extension, the Linked Batch interface type. Batch interfaces are used to manipulate a collection of sub-resources at the same time. Contrary to the basic Batch, which is a collection statically defined by the web server, a Linked Batch is dynamically controlled by a web client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC5988] and the CoRE Link Format [RFC6690]. The draft does not foresee any way to manipulate resources that are located on multiple smart objects with a single client request.

The current CoRE drafts do not foresee any unicast-based way to manipulate resources that are located on multiple nodes with a single client request. To overcome this shortcoming and be able to perform such composite requests, intelligence is typically added to the client application to make it communicate with the nodes individually. This leads to more complex user applications, and the added intelligence and programming cannot be shared with other applications easily. Furthermore, complex use applications may be unmanageable. Any modifications to those complex user applications may require significant testing time, thus limiting the flexibility of the user applications. Additionally a large overhead of communication between the client machine and the nodes is generated, especially when many nodes are involved in these actions. When the communication between the client and the nodes is done across the Internet, delays are unpredictable and a sequence of actuator commands might arrive out of order and possibly have unwanted results. Furthermore, if the communication occurs over costly links, communication between the client and the nodes might get unnecessarily expensive.

The discussed approaches are able to realize communication with a group of resources, but each exhibit some limitations. Therefore, in this Internet Draft we propose an alternative unicast-based approach for communication with a group of resources across multiple nodes.

3. System Overview

We call the component that manages the entities, the Entity Manager (EM). This component, which can reside e.g. on the Border Gateway of the LLN, is responsible for maintaining entities that are created from groups of CoAP servers (i.e. sensors and actuators) and/or resources inside the LLN. Clients on the Internet can interact with an EM to create new entities and/or customize how these entities should behave. Optionally the client can elect to contact a resource directory [I-D.ietf-core-resource-directory] in order to discover which resources are available in the network.

The EM functionality does not have to be put on a dedicated device. Theoretically any CoAP server can be extended to become an EM. The choice of the most appropriate location to put the EM functionality depends on the size and topology of the network. For example, it can reside on a smart object in the constrained network with enough resources, in the Cloud, on the client device itself, or on a gateway at the edge of the LLN. The latter case has the added benefit that security can be centrally managed besides offloading the processing from constrained devices.

Regardless of the location of the EM, it will serve as a proxy between the client and the constrained devices. Client requests will be sent to the EM, which will analyze and verify the requests and then issue the appropriate requests to the constrained devices using CoAP. Once the EM receives responses from the constrained devices, it will combine them according to the client needs and will send back an aggregated response to the client.

When a client tries to create a new entity consisting of a group of resources inside LLNs, the EM performs a sanity check on the request in order to make sure that the resulting entity would make sense. For example it verifies that the resources inside the entity are valid, if they support a certain content format or if their data can be aggregated. Customization of the entity behavior is accomplished by creating profiles for the entities. A profile of an entity can specify for example whether to return the values of all resources in the entity, only the computed average of all values or a subset of all values.

4. Entity Creation

To facilitate the creation and manipulation of entities, an Entity Manager MUST implement the RESTful interface defined in this draft. A CoAP resource implementing this interface can be identified by using the resource type (rt) "core.em". We call this interface the Entity Management Interface and the corresponding resource the Entity Management Resource (available at e.g. "/e"). This interface supports only the CoAP POST request method. As payload of the request, it expects a collection of resources in CoRE link format [RFC6690], which together should form the entity. In the response, the Location-Path CoAP option MUST be used to specify the name of the newly created resource. The payload of the response is in plain text and describes the results of the validation tests performed by the EM on the collection of resources.

When a client wants to create an entity consisting of several sub-resources, it MUST compose a CoAP POST request and send it to the Entity Management resource on the EM. The EM creates the entity, assigns it a unique URI, and stores the entity in the entity database for future usage. Then the EM starts the entity validation process (explained in the next subsection). The EM MUST inform the client about the URI to use in order to access or further customize the newly created entity and about the results of the validation of the entity. If the entity did not pass the validation process the client SHOULD fix any errors and resubmit the entity for validation again before the client can use the entity.

5. Validation Process

Whenever a client requests to create a new entity or to modify an existing entity, the EM SHOULD perform a validation process. The purpose of this validation process is twofold: 1) Make sure that the sub-resources in the entity exist and can be used. 2) Derive the properties of the entity based on the properties of the sub-resources it contains. If the entity passes validation the EM marks the entity as a valid entity and stores the entity along with its calculated properties in the entity database for future usage. If the entity fails validation it is still created, but marked as invalid. The entity validation is based on EM knowledge of the individual sub-resources through .well-known/core and their profiles and possibly based on additional functionality implemented by the EM (e.g. vendor-specific functionality).

If the Entity Manager does not know any of the subresources in an entity (e.g. based on knowledge in a resource directory) or does not know the sub-resource capabilities, it tries to obtain this information according to a fallback mechanism as follows.

- o The EM tries to contact the object containing the resource in order to obtain the resource profile, since this would provide the most complete information about the resource.

- o If the resource profile does not exist, the EM tries to obtain any information about this resource from .well-known/core of the respective object.

- o If this fails as well, the EM tries to query the resource directly to discover, at a minimum, if the resource exists or not.

The validation process that the entity manger performs on entities MUST ensure the following:

- o The individual sub-resources contained in the entity are valid (e.g. the resources exist on the respective nodes).

- o The requested operations can be performed on the individual sub-resources (e.g. which CoAP options are supported, which RESTful methods are allowed?).

- o The individual sub-resources do not conflict. A sample conflict can occur when an entity creation request contains two sub-resources on the same actuator asking it to do contradictory actions, e.g. open and close at the same time.

- o The responses sent by the individual sub-resources can be combined together by using a common denominator or by executing custom algorithms that reside at the EM.

6. Entity Profile

Once the EM knows all information about the subresources that should become part of the entity and once all necessary checks have passed, the EM SHOULD create a profile for the entity based on this information and its custom algorithms. This profile contains information related to the resource itself, as described in [I-D.greevenbosch-core-profile-description]. In addition, the profile is extended with an entity specific part, providing more information about the entity itself. The entity specific part is a JSON object with the name "entity". The value of this object is an array of entity specific fields.

6.1. The resources "r" entity field

The resources "r" entity field contains a list of the resources in the entity. It has the format depicted in Figure 1, where r1, r2, ... are strings containing the absolute URIs of the individual resources.

```
"r":[r1,r2,...]
```

Figure 1: resources "r" entity field syntax

When including the "r" entity field in the entity profile description, all individual resources of the CoAP entity MUST be included.

If the "r" entity profile field is available, the receiving party SHALL assume a non-listed URI is not a resource of the entity.

7. Entity Usage

Once an entity is created the response contains the URI of the dynamically created resource name. The client CAN now interact with the entity by issuing a single CoAP request to the resource representing the entity. When a request for an entity arrives, the following process flow SHOULD be executed.

- o The EM breaks down the request into its components and sends the individual requests to the respective objects using unicast CoAP messages. It can either do that in parallel or sequentially.

- o Once all needed answers are received, the EM creates a response for the client based on the individual responses and sends it to the client. Note that the amount of sub-resources that should respond, the way in which a response is formed and how it should look like can be configured by customizing the entity profile as will be explained later on.

8. Examples

8.1. Entity Creation

In the following simple example the client requests the creation of an entity consisting of two sub-resources: coap://sen5.example.com/tmp and coap://sen8.example.com/tmp. The EM creates the new entity, assigns it the URI "/1" and informs the client about the newly created entity. From now on, any client can access the newly created entity by accessing the "/1" resource on the EM.

```
Req: POST coap://em.example.com/e (application/link-format)
     Body: <coap://sen5.example.com/tmp>,
           <coap://sen8.example.com/tmp>

Res: 2.05 Content (text/plain)
     Body: /1 created
```

8.2. Entity Profile

Assume that the temperature sensor at "coap://sen5.example.com/tmp" from the previous example supports the "Uri-Host" (3), "ETag" (4), "Observe" (6), "Uri-Port" (7), "Uri-Path" (11) and "Content-Format" (12) CoAP options (op). This sensor further supports the "application/senml+json" (55) content format (cf) and the allowed method is GET (1). This will result in Sen5 having the following profile [I-D.greevenbosch-core-profile-description]:

```
Req: GET coap://sen5.example.com/.well-known/profile?path=/tmp

Res: 2.05 Content (application/json)
Body:
{
  "profile":[
    {
      "path":"tmp",
      "op":[3,4,6,7,11,12],
      "cf":[55],
      "m":[1]
    }
  ]
}
```

Let us further assume that the second temperature sensor "coap://sen8.example.com/tmp" supports the same options as sen5 except for the observe option. Only the GET method is allowed and the supported content formats on this sensor are "text/plain" (0) and "application/senml+json" (55). Thus Sen8 will have the following profile:

```
Req: GET coap://sen8.example.com/.well-known/profile?path=/tmp

Res: 2.05 Content (application/json)
Body:
{
  "profile":[
    {
      "path":"tmp",
      "op":[3,4,7,11,12],
      "cf":[0,55],
      "m":[1]
    }
  ]
}
```

Based on these two profiles the EM constructs a profile for the newly created entity. This profile contains information related to the resource itself. In this example, this includes the options that are supported, the supported methods (only GET) and the content format "application/senml+json" (55). In addition, the profile is extended with an entity specific part, providing more information about the entity itself. The resulting profile of the entity looks as follows:

```
Res: 2.05 Content (application/json)
{
  "profile":[
    {
      "path":"1",
      "op":[3,4,7,11,12],
      "cf":[55],
      "m":[1]
    }
  ],
  "entity":[
   {
      "r":["coap://sen5.example.com/tmp",
           "coap://sen8.example.com/tmp"]
   }
  ]
}
```

8.3. Entity Usage

The following Figure shows an example of using the entity that was created previous example. The client issues a GET request on the entity’s resource "/1". This results in the EM issuing two GET requests to the individual sub-resources, waiting for replies from both of them and then sending back both results in one combined response back to the client.

```
Client                          EM                Sen5              Sen8
|                              |                     |                  |
| GET                          |                     |                  |
| coap://em.example.com/1      |                     |                  |
|----------------------------->| GET                 |                  |
|                              | coap://sen5.example.com/tmp            |
|                              |-------------------->|                  |
|                              |                     |                  |
|                              |    GET coap://sen8.example.com/tmp     |
|                              |--------------------------------------->|
|                              |                     |                  |
|                              |<---------------------------------------|
|                         | 2.05 Content (text/plain) Body: 23.5    |
|                         |                     |                   |
|                         |<--------------------|                   |
|                         | 2.05 Content (text/plain) Body: 26.6    |
|                         |                     |                   |
|<------------------------|                     |                   |
| 2.05 Content (application/senml+json)         |                   |
| Payload: {"e":[         |                     |                   |
| {"n": "Sen5/tmp", "v": "26.6", u="degC"},     |                   |
| {"n": "Sen8/tmp", "v": "23.5", u="degC"}]}    |                   |
|                         |                     |                   |
```

9. Open topics

9.1. Open since v00

- Use key words consistently.

10. Security Considerations

For general CoAP security considerations see [I-D.ietf-core-coap].

A client might request the creation of a large number of entities or entities that contain a large number of resources. This might lead to buffer overflow on the EM.

In an unprotected environment, an attacker can change the profile description of Entities. For example, the list of supported options may be changed. This could cause the client to make a wrong decision on which mechanisms to use. As the Entity Manager amplifies a single requests into multiple requests per user, special care should be taken to avoid congestion and to avoid abuse of this mechanism by a malicious user that might want to flood the LLN. However, such threats are normal in environments that lack authentication.

11. IANA Considerations

- A registry for entity profile fields as well as possible values needs to be set up.

12. References

12.1. Normative References

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

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-17 (work in progress), May 2013.

[I-D.greevenbosch-core-profile-description] Greevenbosch, B., Hoebeke, J., and I. Ishaq, "CoAP Profile Description Format", draft-greevenbosch-core-profile-description-01 (work in progress), October 2012.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

12.2. Informative reference

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[I-D.ietf-core-groupcomm] Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-09 (work in progress), May 2013.

[I-D.ietf-core-interfaces] Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-core-interfaces-00 (work in progress), June 2013.

[I-D.ietf-core-resource-directory] Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-ietf-core-resource-directory-00 (work in progress), June 2013.

[I-D.keoh-tls-multicast-security] Keoh, S., Kumar, S., and E. Dijk, "DTLS-based Multicast Security for Low-Power and Lossy Networks (LLNs)", draft-keoh-tls-multicast-security-00 (work in progress), October 2012.

Authors' Addresses

Isam Ishaq
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: +32-9-3314946
Email: isam.ishaq@intec.ugent.be

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

Floris Van den Abeele
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: +32-9-3314946
Email: floris.vandenabeele@intec.ugent.be