

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 10, 2013

J. Arkko  
A. Eriksson  
A. Keranen  
Ericsson  
July 9, 2012

Building Power-Efficient CoAP Devices for Cellular Networks  
draft-arkko-core-cellular-00

Abstract

This memo discusses the use of the Constrained Application Protocol (CoAP) protocol in building sensors and other devices that employ cellular networks as a communications medium. Building communicating devices that employ these networks is obviously well known, but this memo focuses specifically on techniques necessary to minimize power consumption.

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 January 10, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .                    | 3  |
| 2. Goals for Low-Power Operation . . . . .   | 4  |
| 3. Link-Layer Assumptions . . . . .          | 7  |
| 4. Scenarios . . . . .                       | 9  |
| 5. Discovery and Registration . . . . .      | 9  |
| 6. Data Formats . . . . .                    | 11 |
| 7. Real-Time Reachable Devices . . . . .     | 11 |
| 8. Sleepy Devices . . . . .                  | 12 |
| 8.1. Implementation Considerations . . . . . | 13 |
| 9. Security Considerations . . . . .         | 14 |
| 10. IANA Considerations . . . . .            | 14 |
| 11. References . . . . .                     | 15 |
| 11.1. Normative References . . . . .         | 15 |
| 11.2. Informative References . . . . .       | 15 |
| Appendix A. Acknowledgments . . . . .        | 16 |
| Authors' Addresses . . . . .                 | 16 |

## 1. Introduction

This memo discusses the use of the Constrained Application Protocol (CoAP) protocol [I-D.ietf-core-coap] in building sensors and other devices that employ cellular networks as a communications medium. Building communicating devices that employ these networks is obviously well known, but this memo focuses specifically on techniques necessary to minimize power consumption. CoAP has many advantages, including being simple to implement; a thousand lines for the entire software above IP layer is plenty for a CoAP-based sensor, for instance. However, while many of these advantages are obvious and easily obtained, optimizing power consumption remains challenging and requires careful design [I-D.arkko-core-sleepy-sensors].

The memo targets primarily 3GPP cellular networks in their 2G, 3G, and LTE variants and their future enhancements, including possible power efficiency improvements at the radio and link layers. The exact standards or details of the link layer or radios are not relevant for our purposes, however. To be more precise, the material in this memo is suitable for any large-scale, public network that employs point-to-point communications model and radio technology.

Our focus is devices that need to be optimized for power usage, and on devices that employ CoAP. As a general technology, CoAP is similar to HTTP. It can be used in various ways and network entities may take on different roles. This freedom allows the technology to be used in efficient and less efficient ways. Some guidance is needed to understand what communication models over CoAP are recommended when low power usage is a critical goal.

The recommendations in this memo should be taken as complementary to device hardware optimization, microelectronics improvements, and further evolution of the underlying link and radio layers. Further gains in power efficiency can certainly be gained on several fronts; the approach that we take in this memo is to do what can be done at the IP, transport, and application layers to provide the best possible power efficiency. Application implementors generally have to use the current generation microelectronics, currently available radio networks and standards, and so on. This focus in our memo should by no means be taken as an indication that further evolution in these other areas is unnecessary. Such evolution is useful, is ongoing, and is generally complementary to the techniques presented in this memo. The evolution of underlying technologies may change what techniques described here are useful for a particular application, however.

The rest of this memo is structured as follows. Section 2 discusses the need and goals for low-power devices. Section 3 outlines our

expectations for the low layer communications model. Section 4 describes the two scenarios that we address, and Section 5, Section 6, Section 7 and Section 8 give guidelines for use of CoAP in these scenarios.

## 2. Goals for Low-Power Operation

There are many situations where power usage optimization is unnecessary. Optimization may not be necessary on devices that can run on power feed over wired communications media, such as in Power-over-Ethernet (PoE) solutions. These devices may require a rudimentary level of power optimization techniques just to avoid keep overall energy costs and aggregate power feed sizes at a reasonable level, but more extreme techniques necessary for battery powered devices are not required. The situation is similar with devices that can be easily be connected to mains power. Other types of devices may get an occasional charge of power from energy harvesting techniques. For instance, some environmental sensors can run on solar cells. Typically, these devices still have to regulate their power usage in a strict manner, for instance to be able to use as small and inexpensive solar cells as possible.

In battery operated devices the power usage is even more important. For instance, one of the authors employs over a hundred different sensor devices in his home network. A majority of these devices are wired and run on PoE, but in most environments this would be impractical because the necessary wires do not exist. The future is in wireless solutions that can cover buildings and other environments without assuming a pre-existing wired infrastructure. In addition, in many cases it is impractical to provide a mains power source. Often there are no power sockets easily available in the locations that the devices need to be in, and even if there were, setting up the wires and power adapters would be more complicated than installing a standalone device without any wires.

Yet, with a large number of devices the battery lifetimes become critical. Cost and practical limits dictate that devices can be largely just bought and left on their own. For instance, with hundred devices, even a ten-year battery lifetime results in a monthly battery change for one device within the network. This may be impractical in many environments. In addition, some devices may be physically difficult to reach for a battery change. Or, a large group of devices -- such as utility meters or environmental sensors -- cannot be economically serviced too often, even if in theory the batteries could be changed.

| POWER SOURCE | SENSOR COMMUNICATION INTERVAL |                            |                 |
|--------------|-------------------------------|----------------------------|-----------------|
|              | Seconds                       | Minutes or Hours           | Days and longer |
| Battery      | Low-power                     | Low-power or<br>Always-off | Always-off      |
| Harvesting   | Low-power                     | Low-power or<br>Always-off | Always-off      |
| Mains        | Always-on                     | Always-on                  | Always-on       |

Figure 1: Power usage strategies for different classes of applications

Many of these situations lead to a requirement for minimizing power usage and/or maximizing battery lifetimes. A summary of the different situations for sensor-type devices is shown in Figure 1 above. Unfortunately, much of our current technology has been built with different objectives in mind. Networked devices that are "always on", gadgets that require humans to recharge it every couple of days, and protocols that have been optimized to maximize throughput rather than conserve resources.

Long battery lifetimes are required for many applications, however. In some cases these lifetimes should be in the order of years or even a decade or longer. Some communication devices already reach multi-year lifetimes, and continuous improvement in low-power electronics and advances in radio technology keep pushing these lifetimes longer. However, it is perhaps fair to say that battery lifetimes are generally too short at present time.

The general strategies for power usage from Figure 1 can be described as follows:

#### Always-on

Under this strategy, there is no reason for extreme measures for power saving. The device can stay on in the usual manner all the time. It may be useful to employ a power-friendly hardware or limit the number of wireless transmissions, CPU speeds, and other aspects for general power saving and cooling needs, but the device can be in the network all the time.

### Always-off

Under this strategy, the device sleeps such long periods at a time that once it wakes up, it makes sense for it to not pretend that it is connected to the network during those times. These devices re-attach to the network as they are woken up. The main optimization goal is to minimize the effort during such re-attachment process and any resulting application communications.

If the device sleeps for long periods of time, the relative increase in energy expenditure during reattachment for infrequent communication may be acceptable.

### Low-power

These devices need to operate on very small amount of power, but still be able to communicate in a relatively frequent basis. This implies that extremely low power solutions needs to be used for the hardware, chosen link layer mechanisms, and so on. Typically, given the small amount of time between transmissions, despite their sleep state these devices retain some form of network attachment to the network. Techniques used for minimizing power usage for the network communications include minimizing any work from re-establishing communications after waking up, tuning the communications frequency, and paging frequency and other parameters appropriately.

Power usage can not be evaluated solely based on lower layer communications. The entire system, including upper layer protocols and applications is responsible for the power consumption as a whole. The lower communication layers have already adopted many techniques that can be used to reduce power usage, such as scheduling device wake-up times. Further reductions will likely need some co-operation from the upper layers so that unnecessary communications, denial-of-service attacks on power consumptions, and other power drains are eliminated.

Of course, application requirements ultimately determine what kinds of communications are necessary. For instance, some applications require more data to be sent than others. The purpose of the guidelines in this memo is not to prefer one or the other application, but to provide guidance on how to minimize the amount of communications overhead that is not directly required by the application. While such optimization is generally useful, it is relatively speaking most noticeable in applications that transfer only a small amount of data, or operate only infrequently.

### 3. Link-Layer Assumptions

We assume that the underlying communications network can be any large-scale, public network that employs point-to-point communications model and radio technology. 2G, 3G, and LTE networks are examples of such networks, but not the only possible networks with these characteristics.

In the following we look at some of these characteristics and their implications. Note that in most cases these characteristics are not properties of the specific networks but rather inherent in the concept of public networks.

#### Public networks

Using a public network service implies that applications can be deployed without having to build a network to go with them. For economical reasons, only the largest users (such as utility companies) could afford to build their own network, and even they would not be able to provide a world-wide coverage. This means that applications where coverage is important can be built. For instance, most transport sector applications require national or even world-wide coverage to work.

But there are other implications, as well. By definition, the network is not tailored for this application and with some exceptions, the traffic passes through the Internet. One implication of this is that there are generally no application-specific network configurations or discovery support. For instance, the public network helps devices to get on the Internet, set up default routers, configure DNS servers, and so on, but does nothing for configuring possible higher-layer functions, such as servers the device might need to contact to perform its application functions.

Public networks often provide web proxies, and these can in some cases make a significant improvement for delays and cost of communication over the wireless link. For instance, collecting content from a large number of servers used to render a web page and resolving their DNS names in a proxy instead of the user's device may cut down on the general chattiness of the communications, therefore reducing overall delay in completing the entire transaction. However, as of today such proxies are provided only for HTTP communications, not for CoAP.

Similarly, given the lack of available IPv4 addresses, the chances are that many devices are behind a network address translation (NAT) device. This means that they are not easily reachable as

servers. Alternatively, the devices may be directly on the global Internet (either on IPv4 or IPv6) and easily reachable as servers. Unfortunately, this may mean that they also receive unwanted traffic, which may have implications for both power consumption and service costs.

#### Point-to-point link model

This is a common link model in cellular networks. One implication of this model is that there will be no other nodes on the same link, except maybe for the service provider's router. As a result, multicast discovery can not be reasonably used for any local discovery purposes. While the configuration of the service provider's router for specific users is theoretically possible, in practice this is difficult to achieve, at least for any small user that can not afford a network-wide contract for a private APN. The public network access service has little per-user tailoring.

#### Radio technology

The use of radio technology means that power is needed to operate the radios. Transmission generally requires more power than reception. However, radio protocols have generally been designed so that a device checks periodically whether it has messages. In a situation where messages arrive seldomly or not at all, this checking consumes energy. Research has shown that these periodic checks (such as LTE paging message reception) are often a far bigger contributor to energy consumption than message transmission.

Note that for situations where there are several applications on the same device wishing to communicate with the Internet in some manner, bundling those applications together at the same time can be very useful. Some guidance for these techniques in the smartphone context can be found in [Android-Bundle].

Naturally, each device has a freedom to decide when it sends messages. In addition, we assume that there is some way for the devices to control when or how often it wants to receive messages. Specific methods for doing this depend on the specific network being used and also tend to change as improvements in the design of these networks are incorporated. The reception control methods generally come in two variants, fine grained mechanisms that deal with how often the device needs to wake-up for paging messages, and more crude mechanisms where the device simply disconnects from the network for a period of time. There are associated costs and benefits to each method, but those are not relevant for this memo, as long as some control method exists.



#### 4. Scenarios

Not all applications or situations are equal. They may require different solutions or communication models. This memo focuses on two common scenarios:

##### Real-Time Reachable Devices

This scenario involves all communication that requires real-time or near real-time communications with a device. That is, a network entity must be able to reach the device with a small time lag at any time, and no pre-agreed wake-up schedule can be arranged. By "real-time" we mean any reasonable end-to-end communications latency, be it measured in milliseconds or seconds. However, unpredictable sleep states are not expected.

Examples of devices in this category include sensors that must be measurable from a remote source at any instant in time, such as process automation sensors and actuators that require immediate action, such as lightbulbs or door locks.

##### Sleepy Devices

This scenario involves freedom to choose when device communicates. The device is often expected to be able to be in a sleep state for much of its time. The device itself can choose when it communicates, or it lets the network assist in this task.

Examples of devices in this category include sensors that track slowly changing values, such as temperature sensors and actuators that control a relatively slow process, such as heating systems.

Note that there may be hard real-time requirements, but they are expressed in terms of how fast the device can communicate, not in terms of how fast it can respond to a network stimuli. For instance, a fire detector can be classified as a sleepy device as long as it can internally quickly wake up on detecting fire and initiate the necessary communications without delay.

#### 5. Discovery and Registration

In both scenarios the device will be attached to a public network. Without special arrangements, the device will also get a dynamically assigned IP address or an IPv6 prefix. At least one but typically several router hops separate the device from its communicating peers such as application servers. As a result, the address or even the existence of the device is typically not immediately obvious to the

other nodes participating in the application. As discussed earlier, multicast discovery has limited value in public networks; network nodes cannot practically discover individual devices in a large public network. And the devices can not discover who they need to talk, as the public network offers just basic Internet connectivity.

Our recommendation is to initiate a discovery and registration process. This allows each device to inform its peers that it has connected to the network and that it is reachable at a given IP address.

The registration part is easy; a resource directory or mirror proxy can be used. The device should perform the necessary registration with these devices, for instance, as specified in [I-D.shelby-core-resource-directory] and [I-D.vial-core-mirror-proxy]. In order to do this registration, the device needs to know its CORE Link Format description, as specified in [I-D.ietf-core-link-format]. In essence, the registration process involves performing a GET on `.well-known/core/?rt=core-rd` at the address of the resource directory (or `rt=core-mp` for mirror proxies), and then doing a POST on the path of the discovered resource.

However, current CoAP specifications provide limited support for discovering the resource directory or mirror proxy. Local multicast discovery only works in LAN-type networks, but not in these public cellular networks. Our recommended alternate methods for discovery are the following:

#### Manual Configuration

The DNS name of the resource directory or mirror proxy is manually configured. This approach is suitable in situations where the owner of the devices has the resources and capabilities to do the configuration. For instance, a utility company can typically program its metering devices to point to the company servers.

#### Manufacturer Server

The DNS name of the directory or proxy is hardwired to the software by the manufacturer, and the directory or proxy is actually run by the manufacturer. This approach is suitable in many consumer usage scenarios, where it would be unreasonable to assume that the consumer runs any specific network services. The manufacturer's web interface and the directory/proxy servers can co-operate to provide the desired functionality to the end user. For instance, the end user can register a device identity in the manufacturer's web interface and ask specific actions to be taken when the device does something.

### Delegating Manufacturer Server

The DNS name of the directory or proxy is hardwired to the software by the manufacturer, but this directory or proxy merely redirects the request to a directory or proxy run by the whoever bought the device. This approach is suitable in many enterprise environments, as it allows the enterprise to be in charge of actual data collection and device registries; only the initial bootstrap goes through the manufacturer. In many cases there are even legal requirements (such as EU privacy laws) that prevent providing unnecessary information to third parties.

### Common Global Resolution Infrastructure

The delegating manufacturer server model could be generalized into a reverse-DNS -like discovery infrastructure that could answer the question "this is device with identity ID, where is my home registration server?". However, at present no such resolution system exists. (Note: The EPCGlobal system for RFID resolution is reminiscent of this approach.)

## 6. Data Formats

A variety of data formats exist for passing around data. These data formats include XML, JSON, EXI, and text formats. Message lengths can have a significant effect on the amount of energy required for the communications, and such it is highly desirable to keep message lengths minimal. At the same time, extreme optimization can affect flexibility and ease of programming. The authors recommend [I-D.jennings-senml] as a compact, yet easily processed and extendable textual format.

## 7. Real-Time Reachable Devices

These devices are often best modeled as CoAP servers. The device will have limited control on when it receives messages, and it will have to listen actively for messages, up to the limits of the underlying link layer. If the device acts also in client role in some phase of its operation, it can control how many transmissions it makes on its own behalf.

The packet reception checks should be tailored according to the requirements of the application. If sub-second response time is not needed, a slightly more infrequent checking process may save some power.

For sensor-type devices, the CoAP OBSERVE extension [I-D.ietf-core-observe] may be supported. This allows the sensor to track changes to the sensed value, and make an immediate observation response upon a change. This may reduce the amount of polling needed to be done by the client. Unfortunately, it does not reduce the time that the device needs to be listening for requests. Subscription requests from other clients than the currently registered one may come at any time, the current client may change its request, and the device still needs to respond to normal queries as a server. As a result, the sensor can not rely having to communicate only on its own choice of observation interval.

In order to act as a server, the device needs to be placed in a public IPv4 address, be reachable over IPv6, or hosted in a private network. If the the device is hosted on a private network, then all other nodes need to access this device also need to reside in the same private network. There are multiple ways to provide private networks over public cellular networks. One approach is to dedicate a special Access Point Name or APN for the private network. Corporate access via cellular networks has often been arranged in this manner, for instance. Another approach is to use Virtual Private Networking (VPN) technology, for instance IPsec-based VPNs.

Power consumption from unwanted traffic is problematic in these devices, unless placed in a private network or protected by a operator-provided firewall service. Devices on an IPv6 network will have some protection through the nature of the  $2^{64}$  address allocation for a single terminal in a 3GPP cellular network; the attackers will be unable to guess the full IP address of the device. However, this protects only the device from processing a packet, but since the network will still deliver the packet to any of the addresses within the assigned 64-bit prefix, packet reception costs are still incurred.

Note that the the VPN approach can not prevent unwanted traffic received at the tunnel endpoint address, and may require keep-alive traffic. Special APNs can solve this issue, but require explicit arrangement with the service provider.

## 8. Sleepy Devices

These devices are best modeled as devices that can delegate queries to some other node. For instance, as mirror proxy clients [I-D.vial-core-mirror-proxy]. When the device initializes itself, it makes a registration of itself in a mirror proxy as described above in Section 5 and then continues to send periodic updates of sensor values.

As a result, the device acts only as a client, not a server, and can shut down all communication channels while it is during its sleeping period. The length of the sleeping period depends on power and application requirements. Some environmental sensors might use a day or a week as the period, while other devices may use a smaller values ranging from minutes to hours.

Other approaches for delegation include CoAP-options described in [I-D.castellani-core-alive] [I-D.fossati-core-publish-monitor-options]. In this memo we use mirror proxies as an example, because of their ability to work with both HTTP and CoAP implementations; but the concepts are similar and the IETF work is still in progress so the final protocol details are yet to be decided.

The ability to shut down communications and act as only a client has four impacts:

- o Radio transmission and reception can be turned off during the sleeping period, reducing power consumption significantly.
- o However, some power and time is consumed by having to re-attach to the network after the end of a sleep period.
- o The window of opportunity for unwanted traffic to arrive is much smaller, as the device is listening for traffic only part of the time. Note that networks may cache packets for some time though. On the other hand, stateful firewalls can effectively remove much of unwanted traffic for client type devices.
- o The device may exist behind a NAT or a firewall without being impacted. Note that "Simple Security" basic IPv6 firewall capability [RFC6092] blocks inbound UDP traffic by default, so just moving to IPv6 is not direct solution to this problem.

For sleepy devices that represent actuators, it is also possible to use the mirror proxy model. The device can make periodic polls to the proxy to determine if a variable has changed.

### 8.1. Implementation Considerations

There are several challenges in implementing sleepy devices. They need hardware that can be put to an appropriate sleep mode but yet awakened when it is time to do something again. This is not always easy in all hardware platforms. It is important to be able to shut down as much of the hardware as possible, preferably down to everything else except a clock circuit. The platform also needs to support re-awakening at suitable time scales, as otherwise the device

needs to be powered up too frequently.

Most commercial cellular modem platforms do not allow applications to suspend the state of the communications stack. Hence, after a power-off period they need to re-establish communications, which takes some amount of time and extra energy.

Implementations should have a coordinated understanding of the state and sleeping schedule. For instance, it makes no sense to keep a CPU powered up, waiting for a message when the lower layer has been told that the next possible paging opportunity is some time away.

The cellular networks have a number of adjustable configuration parameters, such as the maximum used paging interval. Proper setting of these values has an impact on the power consumption of the device, but with the current business practices, such settings are rarely negotiated when the user's subscription is provisioned.

## 9. Security Considerations

There are no particular security aspects with what has been discussed in this memo, except for the ability to delegate queries for a resource to another node. Depending on how this is done, there are obvious security issues which have largely NOT yet been addressed in the relevant Internet Drafts [I-D.vial-core-mirror-proxy] [I-D.castellani-core-alive] [I-D.fossati-core-publish-monitor-options]. However, we point out that in general, security issues in delegation can be solved either through reliance on your local network support nodes (which may be quite reasonable in many environments) or explicit end-to-end security. Explicit end-to-end security through nodes that are awake at different times means in practice end-to-end data object security. We have implemented one such mechanism for sleepy nodes as described in [I-D.aks-crypto-sensors].

The security considerations relating to CoAP [I-D.ietf-core-coap] and the relevant link layers should apply. Note that cellular networks universally employ per-device authentication, integrity protection, and for most of the world, encryption of all their communications. Additional protection of transport sessions is possible through mechanisms described in [I-D.ietf-core-coap] or data objects.

## 10. IANA Considerations

There are no IANA impacts in this memo.

## 11. References

### 11.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-06 (work in progress), May 2011.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-05 (work in progress), March 2012.
- [I-D.vial-core-mirror-proxy]  
Vial, M., "CoRE Mirror Proxy",  
draft-vial-core-mirror-proxy-00 (work in progress),  
March 2012.
- [I-D.shelby-core-resource-directory]  
Shelby, Z. and S. Krco, "CoRE Resource Directory",  
draft-shelby-core-resource-directory-00 (work in  
progress), June 2011.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format",  
draft-ietf-core-link-format-11 (work in progress),  
January 2012.
- [I-D.jennings-senml]  
Jennings, C., "Media Type for Sensor Markup Language  
(SENML)", draft-jennings-senml-05 (work in progress),  
March 2011.

### 11.2. Informative References

- [RFC6092] Woodyatt, J., "Recommended Simple Security Capabilities in  
Customer Premises Equipment (CPE) for Providing  
Residential IPv6 Internet Service", RFC 6092,  
January 2011.
- [I-D.arkko-core-sleepy-sensors]  
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O.  
Novo, "Implementing Tiny COAP Sensors",  
draft-arkko-core-sleepy-sensors-01 (work in progress),  
July 2011.
- [I-D.arkko-core-security-arch]  
Arkko, J. and A. Keranen, "CoAP Security Architecture",

draft-arkko-core-security-arch-00 (work in progress),  
July 2011.

[I-D.aks-crypto-sensors]

Sethi, M., Arkko, J., Keranen, A., and H. Rissanen,  
"Practical Considerations and Implementation Experiences  
in Securing Smart Object Networks",  
draft-aks-crypto-sensors-02 (work in progress),  
March 2012.

[I-D.ietf-jose-json-web-signature]

Jones, M., Bradley, J., and N. Sakimura, "JSON Web  
Signature (JWS)", draft-ietf-jose-json-web-signature-01  
(work in progress), March 2012.

[I-D.castellani-core-alive]

Castellani, A. and S. Loreto, "CoAP Alive Message",  
draft-castellani-core-alive-00 (work in progress),  
March 2012.

[I-D.fossati-core-publish-monitor-options]

Fossati, T., Giacomini, P., and S. Loreto, "Publish and  
Monitor Options for CoAP",  
draft-fossati-core-publish-monitor-options-01 (work in  
progress), March 2012.

[Android-Bundle]

Developer.Android.Com, "Optimizing Downloads for Efficient  
Network Access", Android developer note [http://  
developer.android.com/training/efficient-downloads/  
efficient-network-access.html](http://developer.android.com/training/efficient-downloads/efficient-network-access.html), <[http://  
developer.android.com/training/efficient-downloads/  
efficient-network-access.html](http://developer.android.com/training/efficient-downloads/efficient-network-access.html)>.

## Appendix A. Acknowledgments

The authors would like to thank Zach Shelby, Jan Holler, Salvatore Loreto, Matthew Vial, Thomas Fossati, Mohit Sethi, Jan Melen, Joachim Sachs, Heidi-Maria Rissanen, Sebastien Pierrel, Kumar Balachandran, Muhammad Waqas Mir, Cullen Jennings, Markus Isomaki, Hannes Tschofenig, and Anna Larmo for interesting discussions in this problem space.



Authors' Addresses

Jari Arkko  
Ericsson  
Jorvas 02420  
Finland

Email: jari.arkko@piuha.net

Anders Eriksson  
Ericsson  
Stockholm 164 83  
Sweden

Email: anders.e.eriksson@ericsson.com

Ari Keranen  
Ericsson  
Jorvas 02420  
Finland

Email: ari.keranen@ericsson.com



Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 3, 2018

J. Arkko  
Ericsson  
C. Jennings  
Cisco  
Z. Shelby  
Sensinode  
October 30, 2017

Uniform Resource Names for Device Identifiers  
draft-arkko-core-dev-urn-05

Abstract

This memo describes a new Uniform Resource Name (URN) namespace for hardware device identifiers. A general representation of device identity can be useful in many applications, such as in sensor data streams and storage, or equipment inventories. A URN-based representation can be easily passed along in any application that needs the information.

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 <https://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 May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .                           | 2  |
| 2. Requirements language . . . . .                  | 3  |
| 3. DEV URN Definition . . . . .                     | 3  |
| 4. DEV URN Subtypes . . . . .                       | 5  |
| 4.1. MAC Addresses . . . . .                        | 5  |
| 4.2. 1-Wire Device Identifiers . . . . .            | 6  |
| 4.3. Organization-Defined Identifiers . . . . .     | 6  |
| 5. Examples . . . . .                               | 6  |
| 6. Security Considerations . . . . .                | 7  |
| 7. IANA Considerations . . . . .                    | 8  |
| 8. References . . . . .                             | 8  |
| 8.1. Normative References . . . . .                 | 8  |
| 8.2. Informative References . . . . .               | 9  |
| Appendix A. Changes from Previous Version . . . . . | 11 |
| Appendix B. Acknowledgments . . . . .               | 12 |
| Authors' Addresses . . . . .                        | 12 |

## 1. Introduction

This memo describes a new Uniform Resource Name (URN) [RFC2141] [RFC3406] namespace for hardware device identifiers. A general representation of device identity can be useful in many applications, such as in sensor data streams and storage, or equipment inventories [RFC7252], [I-D.ietf-core-senml]. A URN-based representation can be easily passed along in any application that needs the information, as it fits in protocols mechanisms that are designed to carry URNs [RFC2616], [RFC3261], [RFC7252]. Finally, URNs can also be easily carried and stored in formats such as XML [W3C.REC-xml-19980210] or JSON [I-D.ietf-core-senml] [RFC4627]. Using URNs in these formats is often preferable as they are universally recognized, self-describing, and therefore avoid the need for agreeing to interpret an octet string as a specific form of a MAC address, for instance.

This memo defines identity URN types for situations where no such convenient type already exist. For instance, [RFC6920] defines cryptographic identifiers, [RFC7254] defines International Mobile station Equipment Identity (IMEI) identifiers for use with 3GPP cellular systems, and [I-D.atarius-dispatch-meid-urn] defines Mobile Equipment Identity (MEID) identifiers for use with 3GPP2 cellular systems. Those URN types should be employed when such identities are

transported; this memo does not redefine these identifiers in any way.

Universally Unique Identifier (UUID) URNs [RFC4122] are another alternative way for representing device identifiers, and already support MAC addresses as one of type of an identifier. However, UUIDs can be inconvenient in environments where it is important that the identifiers are as simple as possible and where additional requirements on stable storage, real-time clocks, and identifier length can be prohibitive. UUID-based identifiers are recommended for all general purpose uses when MAC addresses are available as identifiers. The device URN defined in this memo is recommended for constrained environments.

Future device identifier types can extend the device device URN type defined here, or define their own URNs.

Note that long-term stable unique identifiers are problematic for privacy reasons and should be used with care or avoided as described in [RFC7721].

The rest of this memo is organized as follows. Section 3 defines the "DEV" URN type, and Section 4 defines subtypes for IEEE MAC-48, EUI-48 and EUI-64 addresses and 1-wire device identifiers. Section 5 gives examples. Section 6 discusses the security considerations of the new URN type. Finally, Section 7 specifies the IANA registration for the new URN type and sets requirements for subtype allocations within this type.

## 2. Requirements language

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

## 3. DEV URN Definition

Namespace ID: "dev" requested

Registration Information: This is the first registration of this namespace, 2011-08-27.

Registration version number: 1

Registration date: 2011-08-27

Declared registrant of the namespace: IETF and the CORE working group. Should the working group cease to exist, discussion should be directed to the general IETF discussion forums or the IESG.

Declaration of syntactic structure: The identifier is expressed in ASCII (UTF-8) characters and has a hierarchical structure as follows:

```
devurn = "urn:dev:" body componentpart
body = macbody / owbody / orgbody / otherbody
macbody = "mac:" hexstring
owbody = "ow:" hexstring
orgbody = "dn:" number ":" identifier
otherbody = subtype ":" identifier
subtype = ALPHA *(DIGIT / ALPHA)
identifier = 1*unreservednout
unreservednout = ALPHA / DIGIT / "-" / "."
componentpart = [ "_" component [ componentpart ] ]
component = *1(DIGIT / ALPHA)
hexstring = hexbyte /
             hexbyte hexstring
hexbyte = hexdigit hexdigit
hexdigit = DIGIT / hexletter
hexletter = "a" / "b" / "c" / "d" / "e" / "f"
number = *1DIGIT
```

The above Augmented Backus-Naur Form (ABNF) uses the DIGIT and ALPHA rules defined in [RFC5234], which are not repeated here. The rule for unreserved is defined in Section 2.3 of [RFC3986].

The device identity namespace includes three subtypes, and more may be defined in the future as specified in Section 7.

The optional components following the hexstring are strings depicting individual aspects of a device. The specific strings and their semantics are up to the designers of the device, but could be used to refer to specific interfaces or functions within the device.

Relevant ancillary documentation: See Section 4.

Identifier uniqueness considerations: Device identifiers are generally expected to be unique, barring the accidental issue of multiple devices with the same identifiers.

Identifier persistence considerations: This URN type SHOULD only be used for persistent identifiers, such as hardware-based identifiers or cryptographic identifiers based on keys intended for long-term usage.

Process of identifier assignment: The process for identifier assignment is dependent on the used subtype, and documented in the specific subsection under Section 4.

Process for identifier resolution: The device identities are not expected to be globally resolvable. No identity resolution system is expected. Systems may perform local matching of identities to previously seen identities or configured information, however.

Rules for Lexical Equivalence: The lexical equivalence of the DEV URN is defined as an exact and case sensitive string match. Note that the two subtypes defined in this document use only lower case letters, however. Future types might use identifiers that require other encodings that require a more full-blown character set (such as BASE64), however.

Conformance with URN Syntax: The string representation of the device identity URN and of the MEID sub namespace is fully compatible with the URN syntax.

Validation Mechanism: Specific subtypes may be validated through mechanisms discussed in Section 4.

Scope: DEV URN is global in scope.

#### 4. DEV URN Subtypes

##### 4.1. MAC Addresses

DEV URNs of the "mac" subtype are based on the EUI-64 identifier [IEEE.EUI64] derived from a device with a built-in 64-bit EUI-64. The EUI-64 is formed from 24 or 36 bits of organization identifier followed by 40 or 28 bits of device-specific extension identifier assigned by that organization.

In the DEV URN "mac" subtype the hexstring is simply the full EUI-64 identifier represented as a hexadecimal string. It is always exactly 16 characters long.

MAC-48 and EUI-48 identifiers are also supported by the same DEV URN subtype. To convert a MAC-48 address to an EUI-64 identifier, The OUI of the Ethernet address (the first three octets) becomes the organization identifier of the EUI-64 (the first three octets). The fourth and fifth octets of the EUI are set to the fixed value FFFF hexadecimal. The last three octets of the Ethernet address become the last three octets of the EUI-64. The same process is used to convert an EUI-48 identifier, but the fixed value FFFE is used instead.

Identifier assignment for all of these identifiers rests within the IEEE.

#### 4.2. 1-Wire Device Identifiers

The 1-Wire\* system is a device communications bus system designed by Dallas Semiconductor Corporation. 1-Wire devices are identified by a 64-bit identifier that consists of 8 byte family code, 48 bit identifier unique within a family, and 8 bit CRC code [OW].

\*) 1-Wire is a registered trademark.

In DEV URNs with the "ow" subtype the hexstring is a representation of the full 64 bit identifier as a hexadecimal string. It is always exactly 16 characters long. Note that the last two characters represent the 8-bit CRC code. Implementations MAY check the validity of this code.

Family code and identifier assignment for all 1-wire devices rests with the manufacturers.

#### 4.3. Organization-Defined Identifiers

Device identifiers that have only a meaning within an organisation can also be used to represent vendor-specific or experimental identifiers or identifiers designed for use within the context of an organisation. Organisations are identified by the Private Enterprise Number [RFC2578].

#### 5. Examples

The following three examples provide examples of MAC-based, 1-Wire, and Cryptographic identifiers:



```
urn:dev:mac:0024beffffe804ff1      # The MAC address of
                                     # Jari's laptop

urn:dev:ow:10e2073a01080063         # The 1-Wire temperature
                                     # sensor in Jari's
                                     # kitchen

urn:dev:ow:264437f5000000ed_humidity # The laundry sensor's
                                     # humidity part

urn:dev:ow:264437f5000000ed_temperature # The laundry sensor's
                                     # temperature part

urn:dev:org:32473:123456            # Device 123456 in
                                     # the RFC 5612 example
                                     # organisation
```

## 6. Security Considerations

On most devices, the user can display device identifiers. Depending on circumstances, device identifiers may or may not be modified or tampered by the user. An implementation of the DEV URN MUST NOT change these properties from what they were intended. In particular, a device identifier that is intended to be immutable should not become mutable as a part of implementing the DEV URN type. More generally, nothing in this memo should be construed to override what the relevant device specifications have already said about the identifiers.

Other devices in the same network may or may not be able to identify the device. For instance, on Ethernet network, the MAC address of a device is visible to all other devices.

The URNs generated according to the rules defined in this document result in long-term stable unique identifiers for the devices. Such identifiers may have privacy and security implications because they may enable correlating information about a specific device over a long period of time, location tracking, and device specific vulnerability exploitation [RFC7721]. Also, usually there is no easy way to change the identifier. Therefore these identifiers need to be used with care and especially care should be taken avoid leaking them outside of the system that is intended to use the identifiers.

## 7. IANA Considerations

This document requests the registration of a new URN namespace for "DEV", as described in Section 3.

Additional subtypes for DEV URNs can be defined through IETF Review or IESG Approval [RFC5226].

Such allocations are appropriate when there is a new namespace of some type of device identifiers, defined in stable fashion and with a publicly available specification that can be pointed to.

Note that the organisation (Section 4.3) device identifiers can also be used in some cases, at least as a temporary measure. It is preferable, however, that long-term usage of a broadly employed device identifier be registered with IETF rather than used through the organisation device identifier type.

## 8. References

### 8.1. Normative References

- [IEEE.EUI64] IEEE, "Guidelines For 64-bit Global Identifier (EUI-64)", IEEE , unknown year, <<http://standards.ieee.org/db/oui/tutorials/EUI64.html>>.
- [OW] IEEE, "Overview of 1-Wire(R) Technology and Its Use", MAXIM <http://www.maxim-ic.com/app-notes/index.mvp/id/1796>, June 2008, <<http://www.maxim-ic.com/app-notes/index.mvp/id/1796>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2141] Moats, R., "URN Syntax", RFC 2141, DOI 10.17487/RFC2141, May 1997, <<https://www.rfc-editor.org/info/rfc2141>>.
- [RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <<https://www.rfc-editor.org/info/rfc2578>>.

- [RFC3406] Daigle, L., van Gulik, D., Iannella, R., and P. Faltstrom, "Uniform Resource Names (URN) Namespace Definition Mechanisms", RFC 3406, DOI 10.17487/RFC3406, October 2002, <<https://www.rfc-editor.org/info/rfc3406>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <<https://www.rfc-editor.org/info/rfc5226>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.

## 8.2. Informative References

- [I-D.atarius-dispatch-meid-urn]  
Atarius, R., "A Uniform Resource Name Namespace for the Device Identity and the Mobile Equipment Identity (MEID)", draft-atarius-dispatch-meid-urn-13 (work in progress), October 2017.
- [I-D.ietf-core-senml]  
Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Media Types for Sensor Measurement Lists (SenML)", draft-ietf-core-senml-10 (work in progress), July 2017.
- [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, <<https://www.rfc-editor.org/info/rfc2616>>.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.

- [RFC3971] Arkko, J., Ed., Kempf, J., Zill, B., and P. Nikander, "SECure Neighbor Discovery (SEND)", RFC 3971, DOI 10.17487/RFC3971, March 2005, <<https://www.rfc-editor.org/info/rfc3971>>.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, DOI 10.17487/RFC3972, March 2005, <<https://www.rfc-editor.org/info/rfc3972>>.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <<https://www.rfc-editor.org/info/rfc4627>>.
- [RFC5612] Eronen, P. and D. Harrington, "Enterprise Number for Documentation Use", RFC 5612, DOI 10.17487/RFC5612, August 2009, <<https://www.rfc-editor.org/info/rfc5612>>.
- [RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <<https://www.rfc-editor.org/info/rfc6920>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7254] Montemurro, M., Ed., Allen, A., McDonald, D., and P. Gosden, "A Uniform Resource Name Namespace for the Global System for Mobile Communications Association (GSMA) and the International Mobile station Equipment Identity (IMEI)", RFC 7254, DOI 10.17487/RFC7254, May 2014, <<https://www.rfc-editor.org/info/rfc7254>>.
- [RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <<https://www.rfc-editor.org/info/rfc7721>>.

[W3C.REC-xml-19980210]

Sperberg-McQueen, C., Bray, T., and J. Paoli, "XML 1.0 Recommendation", World Wide Web Consortium First Edition REC-xml-19980210, February 1998, <<http://www.w3.org/TR/1998/REC-xml-19980210>>.

#### Appendix A. Changes from Previous Version

Version -05 made a change to the delimiter for parameters within a DEV URN. Given discussions on allowed character sets in SenML [I-D.ietf-core-senml], we would like to suggest that the "\_" character be used instead of ";", to avoid the need to translate DEV URNs in SenML-formatted communications or files. However, this reverses the earlier decision to not use unreserved characters. This also means that device IDs cannot use "\_" characters, and have to employ other characters instead. Feedback on this decision is sought.

Version -05 also introduced local or organisation-specific device identifiers. Organisations are identified by their PEN number (although we considered FQDNs as a potential alternative. The authors believe an organisation-specific device identifier type will make experiments and local use easier, but feedback on this point and the choice of PEN numbers vs. other possible organisation identifiers would be very welcome.

Version -05 also added some discussion of privacy concerns around long-term stable identifiers.

Finally, version -05 clarified the situations when new allocations within the registry of possible device identifier subtypes is appropriate.

Version -04 is a refresh, as the need and interest for this specification has re-emerged. And the editing author has emerged back to actual engineering from the depths of IETF administration.

Version -02 introduced several changes. The biggest change is that with the NI URNs [RFC6920], it was no longer necessary to define cryptographic identifiers in this specification. Another change was that we incorporated a more generic syntax for future extensions; non-hexstring identifiers can now also be supported, if some future device identifiers for some reason would, for instance, use BASE64. As a part of this change, we also changed the component part separator character from '-' to ';' so that the general format of the rest of the URN can employ the unreserved characters [RFC3986].

## Appendix B. Acknowledgments

The authors would like to thank Ari Keranen, Stephen Farrell, Christer Holmberg, Peter Saint-Andre, Wouter Cloetens, and Ahmad Muhanna for interesting discussions in this problem space. We would also like to note prior documents that focused on specific device identifiers, such as [RFC7254] or [I-D.atarius-dispatch-meid-urn].

## Authors' Addresses

Jari Arkko  
Ericsson  
Jorvas 02420  
Finland

Email: jari.arkko@piuha.net

Cullen Jennings  
Cisco  
170 West Tasman Drive  
San Jose, CA 95134  
USA

Phone: +1 408 421-9990  
Email: fluffy@cisco.com

Zach Shelby  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: zach@sensinode.com

CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: August 24, 2017

K. Kuladinithi, Ed.  
ComNets, Hamburg University of Technology  
M. Becker  
Tridonic GmbH & Co KG  
K. Li  
Alibaba Group  
T. Poetsch  
New York University Abu Dhabi  
February 20, 2017

Transport of CoAP over SMS  
draft-becker-core-coap-sms-gprs-06

Abstract

Short Message Service (SMS) of mobile cellular networks is frequently used in Machine-To-Machine (M2M) communications, such as for telematic devices. The service offers small packet sizes and high delays just as other typical low-power and lossy networks (LLNs), i.e. 6LoWPANs. The design of the Constrained Application Protocol (CoAP, RFC7252), that took the limitations of LLNs into account, is thus also applicable to other transports. The adaptation of CoAP to SMS transport mechanisms is described in this document.

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 August 24, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .   | 3  |
| 1.1. Motivation . . . . .   | 3  |
| 1.2. Terminology . . . . .  | 3  |
| 1.3. Requirements Language . . . . .  | 4  |
| 2. Scenarios . . . . .  | 4  |
| 2.1. MO-MT Scenarios . . . . .  | 4  |
| 2.2. MT Scenarios . . . . .   | 4  |
| 2.3. MO Scenarios . . . . .   | 5  |
| 3. Message Exchanges . . . . .  | 6  |
| 3.1. Message Exchange for SMS in a Cellular-To-Cellular<br>Mobile-Originated and Mobile-Terminated Scenario . . . . . | 6  |
| 4. Encoding Schemes of CoAP for SMS transport . . . . .   | 7  |
| 5. Message Size Implementation Considerations . . . . .   | 7  |
| 6. Addressing . . . . .   | 8  |
| 7. Options . . . . .  | 8  |
| 7.1. New Options for mixed IP operation. . . . .  | 8  |
| 8. URI Scheme . . . . .   | 9  |
| 9. Transmission Parameters . . . . .  | 9  |
| 10. Multicast . . . . .   | 9  |
| 11. Security Considerations . . . . .   | 9  |
| 12. IANA Considerations . . . . .   | 10 |
| 12.1. CoAP Option Number . . . . .  | 10 |
| 12.2. URI Scheme Registration . . . . .   | 10 |
| 13. References . . . . .  | 10 |
| 13.1. Normative References . . . . .  | 10 |
| 13.2. Informative References . . . . .  | 11 |
| Appendix A. SMS encoding . . . . .  | 12 |
| A.1. ASCII-optimized SMS encoding . . . . .   | 12 |
| Appendix B. Changelog . . . . .   | 16 |
| Acknowledgements . . . . .  | 17 |
| Contributors . . . . .  | 17 |
| Authors' Addresses . . . . .  | 17 |



## 1. Introduction

This specification details the usage of the Constrained Application Protocol on the Short Message Service (SMS) of mobile cellular networks.

### 1.1. Motivation

In some M2M environments, internet connectivity is not supported by the constrained end-points, but a cellular network connection is supported instead. Internet connectivity might also be switched off for power saving reasons or the cellular coverage does not allow for Internet connectivity. In these situations, SMS will be supported, instead of UDP/IP over General Packet Radio Service (GPRS), High Speed Packet Access (HSPA) or Long Term Evolution (LTE) networks.

In 3GPP, SMS is identified as the transport protocol for small data transmissions (See [ts23\_888] for Key Issue on Machine Type Communication (MTC) Device Trigger and the proposed solutions in Sections 6.2, 6.42, 6.44, 6.48, 6.52, 6.60, and 6.61). In [ts23\_682] 'Architecture Enhancements to facilitate communications with Packet Data Networks and Applications' SMS is at the moment the only Trigger Delivery (Trigger Delivery using T4).

M2M protocols using SMS, e.g. for telematics, are using mostly various diverse proprietary and closed binary protocols with limited publicly available documentation at the moment.

In Open Mobile Alliance (OMA) LightweightM2M technical specification [oma\_lightweightm2m\_ts], SMS is identified as an alternative transport for CoAP messages.

### 1.2. Terminology

This document uses the following terminology:

#### CoAP Server and Client

The terms CoAP Server and CoAP Client are used synonymously to Server and Client as specified in the terminology section of [RFC7252].

#### Mobile Station (MS)

A Mobile Station includes all required user equipment and software that is needed for communication with a mobile network. As defined in [etsi\_ts101\_748].

### 1.3. 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 RFC 2119 [RFC2119].

## 2. Scenarios

Several scenarios are presented first for M2M communications with CoAP over SMS. First Mobile-Originating Mobile-Terminating (MO-MT) scenarios are presented, where both CoAP endpoints are in devices in a cellular network. Next, Mobile-Terminating (MT) scenarios are detailed, where only the CoAP server is in a cellular network. Finally, Mobile-Originating (MO) scenarios where the CoAP client is in the cellular network.

### 2.1. MO-MT Scenarios

Two mobile cellular terminals communicate by exchanging a CoAP Request and Response embedded into short message protocol data units (PDUs) (depicted in Figure 1). Both terminals are connected via a Short Message Service Centre (SMS-C).

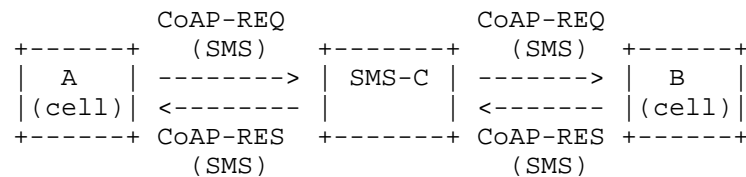


Figure 1: Cellular and Cellular Communication (only SMS-based)

### 2.2. MT Scenarios

An IP host and a mobile cellular terminal communicate by exchanging CoAP Request and Response. The IP host uses protocols offered by the SMS-C (e.g. Short Message Peer-to-Peer (SMPP [smpp]), Computer Interface to Message Distribution (CIMD [cimd]), Universal Computer Protocol/External Machine Interface (UCP/EMI [ucp])) to submit a short message for delivery, which contains the CoAP Request (depicted in Figure 2).

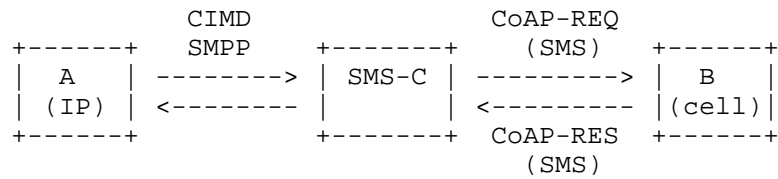


Figure 2: IP and Cellular Communication

There are service providers that offer SMS delivery and notification using an HTTP/REST interface (depicted in Figure 3).

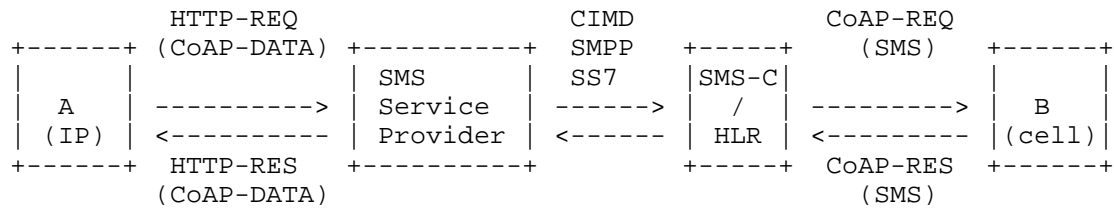


Figure 3: IP and Cellular Communication (using an SMS Service Provider)

### 2.3. MO Scenarios

A mobile cellular terminal and an IP host communicate by exchanging CoAP Request and Response. The mobile cellular terminal sends a CoAP Request in a short message, which is in turn forwarded by the SMS-C (e.g. with Short Message Peer-to-Peer (SMPP [smpp]), Computer Interface to Message Distribution (CIMD [cimd]), Universal Computer Protocol/External Machine Interface (UCP/EMI [ucpl])) as depicted in Figure 4). This scenario can be a fall-back for mobile-originating communication, when IP connectivity cannot be setup (e.g. due to missing coverage).

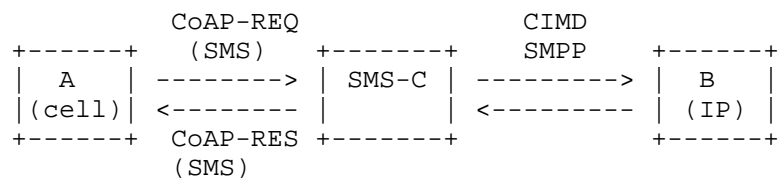


Figure 4: Cellular and IP Communication

There are service providers offering SMS delivery and notification using an HTTP/REST interface (depicted in Figure 5).

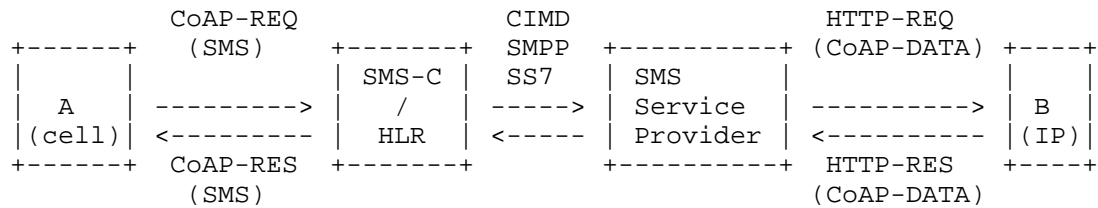


Figure 5: IP and Cellular Communication (using an SMS Service Provider)

### 3. Message Exchanges

#### 3.1. Message Exchange for SMS in a Cellular-To-Cellular Mobile-Originated and Mobile-Terminated Scenario

The CoAP Client works as a Mobile Station to send the short message, and the CoAP Server works as another Mobile Station to receive the short message. All short messages are stored and forwarded by the Service Center. The message exchange between the CoAP Client and the CoAP Server is depicted in the figure below:

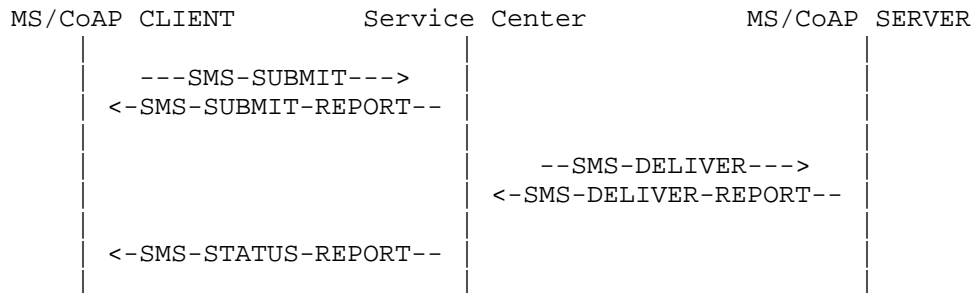


Figure 6: CoAP Messages over SMS

Note that the message exchange is just for one request message from CoAP Client and CoAP Server. It includes the following steps:

Step 1: The CoAP Client sends a CoAP request in a SMS-SUBMIT message to the Service Center. The CoAP Server address is specified as TP-Destination-Address (see [ts23\_040]).

Step 2: The Service Center returns a SMS-SUBMIT-REPORT message to the CoAP Client.

Step 3: The Service Center stores the received SMS message and forwards it to the CoAP Server, using an SMS-DELIVER message. The

CoAP Client address is specified as a TP Originating Address (see [ts23\_040]).

Step 4: The CoAP Server returns an SMS-DELIVER-REPORT message to the Service Center.

Step 5: The Service Center returns the SMS-STATUS-REPORT message to the CoAP Client to indicate the SMS delivery status, if required by the CoAP Client.

Note that the SMS-STATUS-REPORT message just indicates the transport layer SMS delivery status and has no relationship with the confirmable message or non-confirmable message. If the CoAP Client has sent a confirmable message, the CoAP Server MUST use a separate SMS message to transmit the ACK.

#### 4. Encoding Schemes of CoAP for SMS transport

Short messages can be encoded by using various alphabets: GSM 7 bit default alphabet ([ts23\_038]), 8 bit data alphabet, and 16 bit UCS2 data alphabet ([iso\_ucs2]). These encodings lead to message sizes of 160, 140, and 70 characters, respectively. Whereas the support of 7 bit encoding is mandatory on a MS, the two other encodings are dependent on the language that needs to be encoded, e.g. UCS2 for Arabic, Chinese, Japanese, etc. Furthermore, the supported encoding highly depends on the implementations of the MS itself.

According to [ts23\_038], GSM 7 bit encoding shall be supported by all MSs offering SMS services. Since not all MSs support 8 bit short message encoding, the preferred encoding scheme for CoAP messages over SMS is therefore 7 bit, e.g. Base64 ([RFC4648]) or SMS encoding in Appendix A.1.

More considerations about SMS encoding can be found in Appendix A.

#### 5. Message Size Implementation Considerations

By using 7 bit encoding, a maximum length of 160 characters is allowed in one short message [ts23\_038]. Consequently, the maximum length for a CoAP message results in 140 bytes.  $160 \text{ characters} = (140 \text{ bytes} * 8) / 7$ .

Possible options for larger CoAP messages are:

##### Concatenated short messages

Most MSs are able to send concatenation short messages in order to transmit longer messages. The total length of a concatenated short message can consist of up to 255 single messages and result

in total length of 39015 7 bit characters or 34170 bytes.  
 Resulting from this, the maximum length of each individual message reduces to 153 (160 - 7) characters (133 bytes).

#### CoAP block-wise transfer

According to [RFC7959], the Block Size (SZX) of block-wise transfer in CoAP is represented as a three-bit unsigned integer. Thus, the possible block sizes are to the power of two. (Block size =  $2^{(SZX + 4)}$ ). Due to the limitations of 160 characters (140 bytes) for one short message, the maximum value of SZX is 3 (Block size = 128 byte).

However, it is RECOMMENDED that SMS is not used to transfer very large resource data using block-wise transfer.

## 6. Addressing

For SMS in cellular networks, the CoAP endpoints have to work with a SIM (Subscriber Identity Module) card and have to be addressed by the MSISDN (Mobile Station ISDN (MSISDN) number).

To allow the CoAP client to detect that the short message contains a CoAP message, the TP-DATA-Coding-Scheme SHOULD be included.

## 7. Options

### 7.1. New Options for mixed IP operation.

In case a CoAP Server has more than one network interface, e.g. SMS and IP, the CoAP Client might want the server to send the response via an alternative transport, i.e. to its alternative address. However, that implies that the initiating CoAP Client is aware of the presence of the alternative interface. For this reason the new options Response-To-Uri-Host and Response-To-Uri-Port are proposed.

| No. | C | U | N | R | Name                 | Format | Length  | Default |
|-----|---|---|---|---|----------------------|--------|---------|---------|
| TBD |   |   |   |   | Response-To-Uri-Host | string | 1-270 B | (none)  |
| TBD |   |   |   |   | Response-To-Uri-Port | uint   | 0-2 B   | 5683    |

Table 1: New CoAP Option Numbers

If the Response-To-Uri-Host is present in the request, server MUST send the response to the indicated URI address, instead of the client's original request URI.

The options SHOULD NOT be used in the response.

The options MUST NOT occur more than once.

## 8. URI Scheme

The coap:// scheme defines that a CoAP server is reachable over UDP/IP. Hence, a new URI scheme is needed for CoAP servers which are reachable over SMS.

As proposed in [I-D.silverajan-core-coap-alternative-transport], the transport information is expressed as part of the URI scheme component. This is performed by minting new schemes for SMS transport using the form "coap+sms", where the name of the transport is clearly and unambiguously described. The endpoint identifier, path and query components together with each scheme name would be used to uniquely identify each resource.

Example of such URI :

o coap+sms://0015105550101/sensors/temperature

In the URI, 0015105550101 is a telephone subscriber number.

## 9. Transmission Parameters

It is RECOMMENDED to configure the RESPONSE\_TIMEOUT variable for a higher duration than specified in [RFC7252] for the applications described here. The actual value SHOULD be chosen based on experience with SMS.

## 10. Multicast

Multicast is not possible with SMS transports.

## 11. Security Considerations

It is possible that a malicious CoAP Client sends repeated requests, and it may cost money for the CoAP Server to use SMS to send back associated responses. To avoid this situation, the CoAP Server implementation can authenticate the CoAP Client before responding to the requests. For example, the CoAP Server can maintain an MSISDN white list. Only the MSISDN specified in the white list will be

allowed to send requests. The requests from others will be ignored or rejected.

## 12. IANA Considerations

### 12.1. CoAP Option Number

The IANA is requested to add the following option number entries to the CoAP Option Number Registry:

| Number | Name                 | Reference                  |
|--------|----------------------|----------------------------|
| TBD    | Response-To-Uri-Host | Section 2 of this document |
| TBD    | Response-To-Uri-Port | Section 2 of this document |

### 12.2. URI Scheme Registration

According to [I-D.silverajan-core-coap-alternative-transport] this document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+sms". The registration request complies with [RFC7595].

## 13. References

### 13.1. Normative References

- [etsi\_ts101\_748] ETSI, "Technical Report: Digital cellular telecommunications system; Abbreviations and acronyms (GSM 01.04 version 8.0.0 release 1999)", 2000.
- [iso\_ucs2] ISO, "ISO/IEC10646: "Universal Multiple-Octet Coded Character Set (UCS)"; UCS2, 16 bit coding.", 2000.
- [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>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.



- [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>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<http://www.rfc-editor.org/info/rfc7595>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<http://www.rfc-editor.org/info/rfc7959>>.
- [ts23\_038] ETSI 3GPP, "Technical Specification: Alphabets and language-specific information (3GPP TS 23.038 version 11.0.0 Release 11)", 2012.

### 13.2. Informative References

- [cimd] Nokia, "CIMD Interface Specification (SMSCDOC8000.00, Nokia SMS Center 8.0)", 2005.
- [I-D.silverajan-core-coap-alternative-transport] Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transport-09 (work in progress), December 2015.
- [oma\_lightweightm2m\_ts] OMA, "Lightweight Machine to Machine Technical Specification", 2013.
- [RFC1924] Elz, R., "A Compact Representation of IPv6 Addresses", RFC 1924, DOI 10.17487/RFC1924, April 1996, <<http://www.rfc-editor.org/info/rfc1924>>.
- [smpp] SMPP Developers Forum, "Short Message Peer to Peer Protocol Specification v3.4 Issue 1.2", 1999.
- [ts23\_040] 3GPP, "Technical realization of the Short Message Service (SMS)", 3GPP-23.040 a00, March 2011.

- [ts23\_682] ETSI 3GPP, "Technical Specification Group Services and System Aspects; Architecture Enhancements to facilitate communications with Packet Data Networks and Applications; (Release 11)", 2012.
- [ts23\_888] ETSI 3GPP, "Technical Specification Group Services and System Aspects; System Improvements for Machine-Type Communications; (3GPP TR 23.888 version 1.6.0, Release 11)", 2011.
- [ucpl] Vodafone, "Short Message Service Centre (SMSC) External Machine Interface (EMI) Description Version 4.3d", 2011.

## Appendix A. SMS encoding

For use in SMS applications, CoAP messages can be transferred using SMS binary mode. However, there is operational experience showing that some environments cannot successfully send a binary mode SMS.

For transferring SMS in character mode (7-bit characters), base64-encoding [RFC4648] is an obvious choice. 3 bytes of message (24 bits) turn into 4 characters, which consume 28 bits. The overall overhead is approximately 17 %; the maximum message size is 120 bytes (160 SMS characters).

If a more compact encoding is desired, base85 encoding could be employed (however, probably not the version defined in [RFC1924] -- instead, the version used in tools such as btoa and PDF should be chosen). However, this requires division operations. Also, the base85 character set includes several characters that cannot be transferred in a single 7-bit unit in SMS and/or are known to cause operational problems. A modified base85 character set can be defined to solve the latter problem. 4 bytes of message (32 bits) turn into 5 characters, which consume 35 bits. The overall overhead is approximately 9.3 %; the resulting maximum message size is 128 bytes (160 SMS characters).

Base64 and base85 do not make use of the fact that much CoAP data will be ASCII-based. Therefore, we define the following ASCII-optimized SMS encoding.

### A.1. ASCII-optimized SMS encoding

Not all 128 theoretically possible SMS characters are operationally free of problems. We therefore define:

Shunned code characters: @ sign, as it maps to 0x00

LF and CR signs (0x0A, 0x0D)

uppercase C cedilla (0x09), as it is often mistranslated in gateways

ESC (0x1B), as it is used in certain character combinations only

Some ASCII characters cannot be transferred in the base SMS character set, as their code positions are taken by non-ASCII characters. These are simply encoded with their ASCII code positions, e.g., an underscore becomes a section mark (even though underscore has a different code position in the SMS character set).

Equivalently translated input bytes: \$, @, [, \, ], ^, \_, ` , {, |, }, ~, DEL

In other words, bytes 0x20 to 0x7F are encoded into the same code positions in the 7-bit character set.

Out of the remaining code characters, the following SMS characters are available for encoding:

Non-equivalently translated (NET) code characters: 0x01 to 0x08, (8 characters)

0x0B, 0x0C, (2 characters)

0x0E to 0x1A, (13 characters)

0x1C to 0x1F, (4 characters)

Of the 27 NET code characters, 18 are taken as prefix characters (see below), and 8 are defined as directly translated characters:

Directly translated bytes: Equivalently translated input bytes are represented as themselves

0x00 to 0x07 are represented as 0x01 to 0x08

This leaves 0x08 to 0x1F and 0x80 to 0xFF. Of these, the bytes 0x80 to 0x87 and 0xA0 to 0xFF are represented as the bytes 0x00 to 0x07 (represented by characters 0x01 to 0x08) and 0x20 to 0x7F, with a prefix of 1 (see below). The characters 0x08 to 0x1F are represented as the characters 0x28 to 0x3F with a prefix of 2 (see below). The characters 0x88 to 0x9F are represented as the characters 0x48 to 0x5F with a prefix of 2 (see below). (Characters 0x01 to 0x08, 0x20

to 0x27, 0x40 to 0x47, and 0x60 to 0x7f with a prefix of 2 are reserved for future extensions, which could be used for some backreferencing or run-length compression.)

Bytes that do not need a prefix (directly translated bytes) are sent as is. Any byte that does need a prefix (i.e., 1 or 2) is preceded by a prefix character, which provides a prefix for this and the following two bytes as follows:

| char | pfx | . | char | pfx |
|------|-----|---|------|-----|
| 0x0B | 100 | . | 0x15 | 200 |
| 0x0C | 101 | . | 0x16 | 201 |
| 0x0E | 102 | . | 0x17 | 202 |
| 0x0F | 110 | . | 0x18 | 210 |
| 0x10 | 111 | . | 0x19 | 211 |
| 0x11 | 112 | . | 0x1A | 212 |
| 0x12 | 120 | . | 0x1C | 220 |
| 0x13 | 121 | . | 0x1D | 221 |
| 0x14 | 122 | . | 0x1E | 222 |

Table 2: SMS prefix character assignment

(This leaves one non-shunned character, 0x1F, for future extension.)

The coding overhead of this encoding for random bytes is similar to Base85, without the need for a division/multiplication. For bytes that are mostly ASCII characters, the overhead can easily become negative. (Conversely, for bytes that for some reason are more likely to be non-ASCII than in a random sequence of bytes, the overhead becomes greater.)

So, for instance, for the CoAP message in Figure 7:

|              |     |         |       |  |
|--------------|-----|---------|-------|--|
| ver          | tt  | code    | mid   |  |
| 1            | ack | 2.05    | 17033 |  |
| content_type |     | 40      |       |  |
| token        |     | sometok |       |  |

```

3c 2f 3e 3b 74 69 74 6c 65 3d 22 47 65 6e 65 72 |</>;title="Gener
61 6c 20 49 6e 66 6f 22 3b 63 74 3d 30 2c 3c 2f |al Info";ct=0,</
74 69 6d 65 3e 3b 69 66 3d 22 63 6c 6f 63 6b 22 |time>;if="clock"
3b 72 74 3d 22 54 69 63 6b 73 22 3b 74 69 74 6c |;rt="Ticks";titl
65 3d 22 49 6e 74 65 72 6e 61 6c 20 43 6c 6f 63 |e="Internal Cloc
6b 22 3b 63 74 3d 30 2c 3c 2f 61 73 79 6e 63 3e |k";ct=0,</async>
3b 63 74 3d 30 |;ct=0

```

Figure 7: CoAP response message as captured and decoded

The 116 byte unencoded message is shown as ASCII characters in Figure 8 (\xDD stands for the byte with the hex digits DD):

```

bEB\x89\x11(\xA7sometok</>;title="General Info";ct=0,</time>
;if="clock";rt="Ticks";title="Internal Clock";ct=0,</async>;ct=0

```

Figure 8: CoAP response message shown as unencoded characters

The only non-ASCII characters in this example are in the beginning of the message. According to the translation instructions above, the four bytes:

```
89 11 ( A7
```

need the prefixes:

```
2 2 0 1
```

As each prefix character always covers three unencoded bytes, we need the prefix characters for 220 and 100, which are \x1C and \x0B, respectively (Table 2).

The equivalent SMS encoding is shown as equivalent-coded SMS characters in Figure 9 (7 bits per character, \x1C is the 220 prefix and \x0B is the 100 prefix, the rest is shown in equivalent encoding), adding two characters of prefix overhead, for a total length of 118 7-bit characters or 104 (103.25 plus padding) bytes:

```

bEB\x1CI1(\x0B'sometok</>;title="General Info";ct=0,</time>
;if="clock";rt="Ticks";title="Internal Clock";ct=0,</async>;ct=0

```

Figure 9: CoAP response message shown as SMS-encoded characters

## Appendix B. Changelog

RFC editor: please remove this appendix.

Changed from draft-05 to draft-06:

- o Update references and addresses
- o Integrate relevant text from coap-misc as an appendix.
- o Section 2 & 3 are merged to section 1

Changed from draft-04 to draft-05:

- o Removed reference to USSD.
- o Updated reference to RFC7252 and 3GPP specs.
- o Updated Options.
- o Adapted URI scheme.

Changed from draft-03 to draft-04:

- o Removed USSD and GPRS related parts.
- o Removed section 5: Examples
- o Removed section 14: Proxying Considerations
- o Added more block size considerations.
- o Added more concatenated SMS considerations.
- o Rewrote encoding scheme section; 7 bit encoding only.

Changed from draft-02 to draft-03:

- o Added reference to OMA LightweightM2M Technical Specification in "Motivation" section.
- o Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-13.

Changed from draft-01 to draft-02:

- o Added security considerations: Transport and Object Security.  
Section 11

- o Reply-To-\* changed to Response-To-\*. Section 12
- o Added URI scheme.
- o Added possible CON/NON/ACK interactions.
- o Added possible M2M proxy scenarios.
- o Added reference to bormann-coap-misc for other SMS encoding. Section 4
- o Updated requirements on Uri-Host and Uri-Port for coap+tel://.
- o Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-10. />
- o Added an IANA registration for the URI scheme. Section 12.2

#### Acknowledgements

This document is partly based on research for the research project 'The Intelligent Container' which is supported by the Federal Ministry of Education and Research, Germany, under reference number 01IA10001.

The authors of this draft would like to thank Bert Greevenbosch, Marcus Goetting, Nils Schulte and Klaus Hartke for the discussions on the topic and the reviews of this document.

#### Contributors

Appendix A has been contributed by Carsten Bormann.

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
EMail: cabo@tzi.org

#### Authors' Addresses

Koojana Kuladinithi (editor)  
ComNets, Hamburg University of Technology  
Am Schwarzenberg-Campus 3  
Hamburg 21073  
Germany

Phone: +49 40 428 783533  
Email: koojana.kuladinithi@tuhh.de

Markus Becker  
Tridonic GmbH & Co KG  
Faerbergasse 15  
Dornbirn 6851  
Austria

Phone: +43 5572 395 45637  
Email: markus.becker@tridonic.com

Kepeng LI  
Alibaba Group  
Wenyixi Road, Yuhang District  
Hangzhou, Zhejiang 311121  
China

Email: kepeng.lkp@alibaba-inc.com

Thomas Poetsch  
New York University Abu Dhabi  
P.O. Box 129188  
Abu Dhabi 129188  
United Arab Emirates

Phone: +971 2 628 5069  
Email: thomas.poetsch@nyu.edu



CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 17, 2013

C. Bormann  
K. Hartke  
Universitaet Bremen TZI  
July 16, 2012

Miscellaneous additions to CoAP  
draft-bormann-coap-misc-19

Abstract

This short I-D makes a number of partially interrelated proposals how to solve certain problems in the CoRE WG's main protocol, the Constrained Application Protocol (CoAP). The current version has been resubmitted to keep information about these proposals available; the proposals are not all fleshed out at this point in time.

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 January 17, 2013.

Copyright Notice

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

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

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .  | 4  |
| 2. Getting rid of artificial limitations . . . . .                           | 5  |
| 2.1. Option Length encoding beyond 270 bytes . . . . .                       | 5  |
| 3. Registered Option . . . . .   | 8  |
| 3.1. A Separate Suboption Number Space . . . . .                             | 8  |
| 3.2. Opening Up the Option Number Space . . . . .                            | 9  |
| 3.2.1. Long Jump construct . . . . .   | 10 |
| 3.2.2. Discussion . . . . .  | 11 |
| 3.2.3. Example . . . . .   | 12 |
| 3.2.4. IANA considerations . . . . .   | 12 |
| 4. Patience, Leisure, and Pledge . . . . .                                   | 13 |
| 4.1. Patience . . . . .  | 13 |
| 4.2. Leisure . . . . .   | 13 |
| 4.3. Pledge . . . . .  | 14 |
| 4.4. Option Formats . . . . .  | 14 |
| 5. Observing Resources in CoAP . . . . .                                     | 16 |
| 6. CONNECT . . . . .   | 19 |
| 6.1. Requesting a Tunnel with CONNECT . . . . .                              | 19 |
| 6.2. Using a CONNECT Tunnel . . . . .  | 19 |
| 6.3. Closing down a CONNECT Tunnel . . . . .                                 | 20 |
| 7. IANA Considerations . . . . .   | 21 |
| 8. Security Considerations . . . . .   | 22 |
| 9. Acknowledgements . . . . .  | 23 |
| 10. References . . . . .   | 24 |
| 10.1. Normative References . . . . .   | 24 |
| 10.2. Informative References . . . . .                                       | 24 |
| Appendix A. The Nursery (Things that still need to ripen a<br>bit) . . . . . | 26 |
| A.1. Envelope Options . . . . .  | 26 |
| A.2. Payload-Length Option . . . . .   | 27 |
| A.3. URI Authorities with Binary Addresses . . . . .                         | 27 |
| A.4. Length-aware number encoding (o256) . . . . .                           | 28 |
| A.5. SMS encoding . . . . .  | 30 |
| A.5.1. ASCII-optimized SMS encoding . . . . .                                | 31 |
| Appendix B. The Cemetery (Things we won't do) . . . . .                      | 34 |
| B.1. Example envelope option: solving #230 . . . . .                         | 34 |
| B.2. Example envelope option: proxy-elective options . . . . .               | 35 |
| B.3. Stateful URI compression . . . . .                                      | 35 |
| B.4. Beyond 270 bytes in a single option . . . . .                           | 36 |
| B.5. Beyond 15 options . . . . .   | 37 |
| B.5.1. Implementation considerations . . . . .                               | 39 |
| B.5.2. What should we do now? . . . . .                                      | 40 |
| B.5.3. Alternatives . . . . .  | 40 |
| B.5.4. Alternative: Going to a delimiter model . . . . .                     | 40 |
| B.6. Implementing the option delimiter for 15 or more<br>options . . . . .   | 40 |

|   |    |
|---|----|
| Appendix C. Experimental Options . . . . .      | 42 |
| C.1. Options indicating absolute time . . . . . | 42 |
| C.2. Representing Durations . . . . .           | 43 |
| C.3. Rationale . . . . .                        | 44 |
| C.4. Pseudo-Floating Point . . . . .            | 45 |
| C.5. A Duration Type for CoAP . . . . .         | 46 |
| Authors' Addresses . . . . .                    | 53 |

## 1. Introduction

The CoRE WG is tasked with standardizing an Application Protocol for Constrained Networks/Nodes, CoAP [I-D.ietf-core-coap]. This protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC2616], while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

This objective requires restraint in a number of sometimes conflicting ways:

- o reducing implementation complexity in order to minimize code size,
- o reducing message sizes in order to minimize the number of fragments needed for each message (in turn to maximize the probability of delivery of the message), the amount of transmission power needed and the loading of the limited-bandwidth channel,
- o reducing requirements on the environment such as stable storage, good sources of randomness or user interaction capabilities.

This draft attempts to address a number of problems not yet adequately solved in [I-D.ietf-core-coap]. The solutions proposed to these problems are somewhat interrelated and are therefore presented in one draft.

The appendix contains the "CoAP cemetery" (possibly later to move into its own draft), documenting roads that the WG decided not to take, in order to spare readers from reinventing them in vain.

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 term "byte" is used in its now customary sense as a synonym for "octet".

## 2. Getting rid of artificial limitations

Artificial limitations are limitations of a protocol or system that are not rooted in limitations of actual capabilities, but in arbitrary design decisions. Proper system design tries to avoid artificial limitations, as these tend to cause complexity in systems that need to work with these limitations.

E.g., the original UNIX filesystem had an artificial limitation of the length of a path name component to 14 bytes. This led to a cascade of workarounds in programs that manipulate file names: E.g., systematically replacing a ".el" extension in a filename with a ".elc" for the compiled file might exceed the limit, so all ".el" files were suddenly limited to 13-byte filenames.

Note that, today, there still is a limitation in most file system implementations, typically at 255. This just happens to be high enough to rarely be of real-world concern; we will refer to this case as a "painless" artificial limitation.

CoAP-08 had two highly recognizable artificial limitations in its protocol encoding

- o The number of options in a single message is limited to 15 max.
- o The length of an option is limited to 270 max.

It has been argued that the latter limitation causes few problems, just as the 255-byte path name component limitation in filenames today causes few problems. Appendix B.4 provided a design to extend this; as a precaution to future extensions of this kind, the current encoding for length 270 (eight ones in the extension byte) could be marked as reserved today. Since, Matthias Kovatsch has proposed a simpler scheme that seems to gain favor in the WG, see Section 2.1.

The former limitation has been solved in CoAP-09. A historical discussion of other approaches for going beyond 15 options is in Appendix B.5. Appendix B.6 discusses implementation.

### 2.1. Option Length encoding beyond 270 bytes

For option lengths beyond 270 bytes, we reserve the value 255 of an extension byte to mean "add 255, read another extension byte" Figure 1. While this causes the length of the option header to grow linearly with the size of the option value, only 0.4 % of that size is used. With a focus on short options, this encoding is justified.

[illegible]

Figure 1: Options beyond 270 bytes

Options that are longer than 1034 bytes MUST NOT be sent; an option that has 255 (all one bits) in the field called "Length - 780" MUST be rejected upon reception as an invalid option.

In the process, the maximum length of all options that are currently set at 270 should now be set to a carefully chosen value. With the purely encoding-based limit gone, Uri-Proxy should now be restored to be a non-repeatable option.

A first proposal for a new set of per-option length restrictions follows:

| number | name                | min | max  | type   | repeat |
|--------|---------------------|-----|------|--------|--------|
| 1      | content_type        | 0   | 2    | uint   | -      |
| 2      | max_age             | 0   | 4    | uint   | -      |
| 3      | proxy_uri           | 1   | 1023 | string | -      |
| 4      | etag                | 1   | 8    | opaque | yes    |
| 5      | uri_host            | 1   | 255  | string | -      |
| 6      | location_path       | 0   | 255  | string | yes    |
| 7      | uri_port            | 0   | 2    | uint   | -      |
| 8      | location_query      | 0   | 255  | string | yes    |
| 9      | uri_path            | 0   | 255  | string | yes    |
| 10     | observe             | 0   | 2    | uint   | -      |
| 11     | token               | 1   | 8    | opaque | -      |
| 12     | accept              | 0   | 2    | uint   | yes    |
| 13     | if_match            | 0   | 8    | opaque | yes    |
| 14     | registered_elective | 1   | 1023 | opaque | yes    |
| 15     | uri_query           | 1   | 255  | string | yes    |
| 17     | block2              | 0   | 3    | uint   | -      |
| 18     | size                | 0   | 4    | uint   | -      |
| 19     | block1              | 0   | 3    | uint   | -      |
| 21     | if_none_match       | 0   | 0    | empty  | -      |
| 25     | registered_critical | 1   | 1023 | opaque | yes    |

(Option 14 with a length of 0 is a fencepost only.)

### 3. Registered Option

CoAP's option encoding is highly efficient, but works best with small option numbers that do not require much fenceposting. The CoAP Option Number Registry therefore has a relatively heavyweight registration requirement: "IETF Review" as described in [RFC5226].

However, there is also considerable benefit in a much looser registry policy, enabling a first-come-first-served policy for a relatively large option number space.

Here, we discuss two solutions that enable such a registry. One is to define a separate mechanism for registered options, discussed in Section 3.1. Alternatively, we could make it easier to use a larger main option number space, discussed in Section 3.2.

#### 3.1. A Separate Suboption Number Space

This alternative defines a separate space of suboption numbers, with an expert review [RFC5226] (or even first-come-first-served) registration policy. If expert review is selected for this registry, it would be with a relatively loose policy delegated to the expert. This draft proposes leaving the registered suboption numbers 0-127 to expert review with a policy that mainly focuses on the availability of a specification, and 128-16383 for first-come-first-served where essentially only a name is defined.

The "registered" options are used in conjunction with this suboption number registry. They use two normal CoAP option numbers, one for options with elective semantics (Registered-Elective) and one for options with critical semantics (Registered-Critical). The suboption numbers are not separate, i.e. one registered suboption number might have some elective semantics and some other critical semantics (e.g., for the request and the response leg of an exchange). The option value starts with an SDNV [RFC6256] of the registered suboption number. (Note that there is no need for an implementation to understand SDNVs, it can treat the prefixes as opaque. One could consider the SDNVs as a suboption prefix allocation guideline for IANA as opposed to a number encoding.)

```

+-----+
|1 0 0 0 0 0 1|0 1 1 1 0 0 1 1|          value...          |
+-----+
\___SDNV of registered number___/

```

Figure 2: Example option value for registered option

Note that a Registered Option cannot be empty, because there would be



no space for the SDNV. Also, the empty option 14 is reserved for fenceposting ([I-D.ietf-core-coap], section 3.2). (Obviously, once a Registered-Elective Option is in use, there is never a need for a fence-post for option number 14.)

The Registered-Elective and Registered-Critical Options are repeatable.

| No. | C/E      | Name                | Format      | Length      | Default |
|-----|----------|---------------------|-------------|-------------|---------|
| 14  | Elective | Registered-Elective | (see above) | 1-1023<br>B | (none)  |
| 25  | Critical | Registered-Critical | (see above) | 1-1023<br>B | (none)  |

This solves CoRE issue #214 [CoRE214]. (How many options we need will depend on the resolution of #241 [CoRE241].)

### 3.2. Opening Up the Option Number Space

The disadvantage of the registered-... options is that there is a significant syntactic difference between options making use of this space and the usual standard options. This creates a problem not unlike that decried in [RFC6648].

The alternative discussed in this section reduces the distance by opening up the main Option number space instead.

There is still a significant incentive to use low-numbered Options. However, the proposal reduces the penalty for using a high-numbered Option to two or three bytes. More importantly, using a cluster of related high-numbered options only carries a total penalty of two or three bytes.

The main reason high-numbered options are expensive to use and thus the total space is relatively limited is that the option delta mechanism only allows increasing the current option number by up to 14 per one-byte fencepost. To use, e.g., Option number 1234 together with the usual set of low-numbered Options, one needs to insert 88 fence-post bytes. This is prohibitive.

Enabling first-come-first-served probably requires easily addressing a 16-bit option number space, with some potential increase later in the lifetime of the protocol (say, 10 to 15 years from now).

To enable the use of large option numbers, one needs a way to advance the Option number in bigger steps than possible by the Option Delta. So we propose a new construct, the Long Jump construct, to move the Option number forward.

### 3.2.1. Long Jump construct

The following construct can occur in front of any Option:

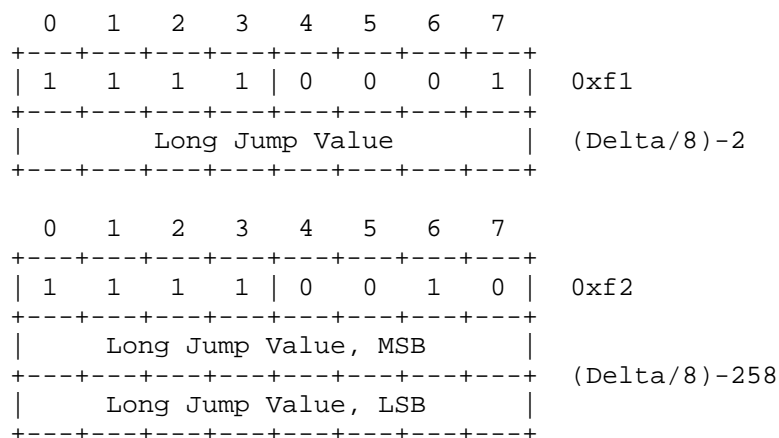


Figure 3: Long Jump Format

This construct is not by itself an Option. It can occur in front of any Option to increase the current Option number that then goes into its Option number calculation. The increase is done in multiples of eight. More specifically, the actual addition to the current Option number is computed as follows:

$$\text{Delta} = ((\text{Long Jump Value}) + N) * 8$$

where N is 2 for the one-byte version and N is 258 for the two-byte version.

A Long Jump MUST be followed by an actual Option, i.e., it MUST NOT be followed by another Long Jump or an end-of-options indicator. A message violating this MUST be rejected as malformed.

Long Jumps do NOT count as Options in the Option Count field of the header (i.e., they cannot by themselves end the Option sequence).

### 3.2.2. Discussion

Adding a mechanism at this late stage creates concerns of backwards compatibility. A message sender never needs to implement long-jumps unless it wants to make use of a high-numbered option. So this mechanism can be added once a high-numbered option is added. A message receiver, though, would more or less unconditionally have to implement the mechanism, leading to unconditional additional complexity. There are good reasons to minimize this, as follows:

- o The increase in multiples of eight allows looking at an option and finding out whether it is critical or not even if the Long Jump value has just been skipped (as opposed to having been processed fully). (It also allows accessing up to approximately 2048 options with a two-byte Long Jump.) This allows a basic implementation that does not implement any high-numbered options to simply ignore long jumps and any elective options behind them, while still properly reacting to critical options.
- o There is probably a good reason to disallow long-jumps that lead to an option number of 42 and less, enabling simple receivers to do the above simplification.
- o It might seem obvious to remove the fenceposting mechanism altogether in favor of long jumps. This is not advisable: Fenceposting already has zero implementation effort at the receiver, and the overhead at the sender is very limited (it is just a third kind of jump, at one byte per jump). Beyond 42, senders can ignore the existence of fenceposts if they want (possibly obviating the need for more complex base-14 arithmetic).

There is no need for a finer granularity than 8, as the Option construct following can also specify a Delta of 0..14. (A granularity of 16 will require additional fenceposting where an option delta of 15 would happen to be required otherwise, which we have reserved. It can be argued that 16 is still the better choice, as fenceposting is already in the code path.)

The Long Jump construct takes 0xf1 and 0xf2 from the space available for initial bytes of Options. (Note that we previously took 0xf0 to indicate end-of-options for OC=15.)

Varying N with the length as defined above makes it unambiguous whether a one- or two-byte Long Jump is to be used. Setting N=2 for the one-byte version makes it clear that a Delta of 8 is to be handled the usual way (i.e., by Option Delta itself and/or fenceposting). If the delta is not small and not 7 modulo 8, there is still a choice between using the smaller multiple of 8 and a

larger Delta in the actual Option or v.v., this biases the choice towards a larger Long Jump and a smaller following Delta, which is also easier to implement as it reduces the number of choice points.

### 3.2.3. Example

The following sequence of bytes would encode a Uri-Path Option of "foo" followed by Options 1357 (value "bar") and 1360 (value "baz"):

```

93 65 6f 6f      Option 9 (0 + 9, "foo")
f1 a6            Long Jump by 1344
43 62 61 72      Option 1357 (9 + 1344 + 4, "bar")
33 62 61 7a      Option 1360 (1357 + 3, "baz")

```

Figure 4: Example using a Long Jump construct

where f1 a6 is the long jump forward by  $(0xa6+2)*8=1344$  option numbers. The total option count (OC) for the CoAP header is 3. Note that even if f1 a6 is skipped, the 1357 (which then appears as an Option number 13) is clearly visible as Critical.

### 3.2.4. IANA considerations

With the scheme proposed above, we could have three tiers of Option Numbers:

| Option Number | Policy [RFC5226]        |
|---------------|-------------------------|
| 0..255        | Standards Action        |
| 256..2047     | Designated Expert       |
| 2048..65535   | First Come First Served |

For the inventor of a new option, this would provide a small incentive to go through the designated expert for some minimal cross-checking in order to be able to use the two-byte long-jump.

#### 4. Patience, Leisure, and Pledge

A number of options might be useful for controlling the timing of interactions.

(This section also addresses core-coap ticket #177.)

##### 4.1. Patience

A client may have a limited time period in which it can actually make use of the response for a request. Using the Patience option, it can provide an (elective) indication how much time it is willing to wait for the response from the server, giving the server license to ignore or reject the request if it cannot fulfill it in this period.

If the server knows early that it cannot fulfill the request in the time requested, it MAY indicate this with a 5.04 "Timeout" response. For non-safe methods (such as PUT, POST, DELETE), the server SHOULD indicate whether it has fulfilled the request by either responding with 5.04 "Timeout" (and not further processing the request) or by processing the request normally.

Note that the value of the Patience option should be chosen such that the client will be able to make use of the result even in the presence of the expected network delays for the request and the response. Similarly, when a proxy receives a request with a Patience option and cannot fulfill that request from its cache, it may want to adjust the value of the option before forwarding it to an upstream server.

(TBD: The various cases that arise when combining Patience with Observe.)

The Patience option is elective. Hence, a client MUST be prepared to receive a normal response even after the chosen Patience period (plus an allowance for network delays) has elapsed.

##### 4.2. Leisure

Servers generally will compute an internal value that we will call Leisure, which indicates the period of time that will be used for responding to a request. A Patience option, if present, can be used as an upper bound for the Leisure. Leisure may be non-zero for congestion control reasons, in particular for responses to multicast requests. For these, the server should have a group size estimate  $G$ , a target rate  $R$  (which both should be chosen conservatively) and an estimated response size  $S$ ; a rough lower bound for Leisure can then be computed as follows:

$$lb\_Leisure = S * G / R$$

Figure 5: Computing a lower bound for the Leisure

E.g., for a multicast request with link-local scope on an 2.4 GHz IEEE 802.15.4 (6LoWPAN) network, G could be (relatively conservatively) set to 100, S to 100 bytes, and the target rate to 8 kbit/s = 1 kB/s. The resulting lower bound for the Leisure is 10 seconds.

To avoid response implosion, responses to multicast requests SHOULD be dithered within a Leisure period chosen by the server to fall between these two bounds.

Currently, we don't foresee a need to signal a value for Leisure from client to server (beyond the signalling provided by Patience) or from server to client, but an appropriate Option might be added later.

#### 4.3. Pledge

In a basic observation relationship [I-D.ietf-core-observe], the server makes a pledge to keep the client in the observation relationship for a resource at least until the max-age for the resource is reached.

To save the client some effort in re-establishing observation relationships each time max-age is reached, the server MAY want to extend its pledge beyond the end of max-age by signalling in a response/notification an additional time period using the Pledge Option, in parallel to the Observe Option.

The Pledge Option MUST NOT be used unless the server can make a reasonable promise not to lose the observation relationship in this time frame.

Currently, we don't foresee a need to signal a value for Pledge from client to server, but an appropriate behavior might be added later for this option when sent in a request.

#### 4.4. Option Formats

| No. | C/E      | Name     | Format         | Length | Default |
|-----|----------|----------|----------------|--------|---------|
| 22  | Elective | Patience | Duration in ms | 1 B    | (none)  |
| 24  | Elective | Pledge   | Duration in s  | 1 B    | 0       |

All timing options use the Duration data type (see Appendix C.2), however Patience (and Leisure, if that ever becomes an option) uses a timebase of mibiseconds (mis = 1/1024 s) instead of seconds. (This reduces the range of the Duration from ~ 91 days to 128 minutes.)

Implementation note: As there are no strong accuracy requirements on the clocks employed, making use of any existing time base of milliseconds is a valid implementation approach (2.4 % off).

None of the options may be repeated.

## 5. Observing Resources in CoAP

(Co-Author for this section: Matthias Kovatsch)

There are two open issues related to -observe  
[I-D.ietf-core-observe]:

- o mixing freshness and observation lifetime, and
- o non-cacheable resources.

To solve the first issue, we think that -observe should be clarified as follows:

A server sends at least some notifications as confirmable messages. Each confirmable notification is an opportunity for the server to check if the client is still there. If the client acknowledges the notification, it is assumed to be well and alive and still interested in the resource. If it rejects the message with a reset message or if it doesn't respond, it is assumed not longer to be interested and is removed from the list of observers. So an observation relationship can potentially go on forever, if the client acknowledges each confirmable notification. If the server doesn't send a notification for a while and wants to check if the client is still there, it may send a confirmable notification with the current resource state to check that.

So there is no mixing of freshness and lifetime going on.

The other issue is a bit less trivial to solve. The problem is that normal CoAP and -observe actually have very different freshness models:

Normally, when a client wants to know the current state of a resource, it retrieves a representation, uses it and stores it in its cache. Later, when it wants to know the current state again, it can either use the stored representation provided that it's still fresh, or retrieve a new representation, use it and store it in its cache.

If a server knows when the state of the resource will change the next time, it can set the Max-Age of the representation to an accurate time span. So the change of the resource state will coincide with the expiration of the freshness of the representation stored in the client's cache (ignoring network latency).

But if the resource changes its state unpredictably at any time, the server can set the Max-Age only to an estimate. If the state then actually changes before the freshness expires, the client wrongly



believes it has fresh information. Conversely, if the freshness expires and the client wants to know the current state, the client wrongly believes it has to make a new request although the representation is actually still fresh (this is defused by ETag validation).

-observe doesn't have these kinds of problems: the server does not have to predict when the resource will change its state the next time. It just sends a notification when it does. The new representation invalidates the old representation stored in the client's cache. So the client always has a fresh representation that it can use when it wants to know the current resource state without ever having to make a request. An explicit Max-Age is not needed for determining freshness.

But -observe has a different set of problems:

The first problem is that the resource may change its state more often than there is bandwidth available or the client can handle. Thus, -observe cannot make any guarantee that a client will see every state change. The solution is that -observe guarantees that the client will eventually see the latest state change, and follows a best effort approach to enable the client to see as many state changes as possible.

The second problem is that, when a notification doesn't arrive for a while, the client does not know if the resource did not change its state or if the server lost its state and forgot that the client is interested in the resource. We propose the following solution: With each notification that the server sends, it makes a promise to send another notification, and that it will send this next notification at latest after a certain time span. This time span is included with each notification. So when no notification arrives for a while and the time span has not expired yet, the client assumes that the resource did not change its state. If the time span has expired, no notification has arrived and the client wants to know the current state of the resource, it has to make a new request.

The third problem is that, when an intermediary is observing a resource and wants to create a response from a representation stored in its cache, it needs to specify a Max-Age. But the intermediary cannot predict when it will receive the next notification, because the next notification can arrive at any time. Unlike the origin server, it also doesn't have the application-specific knowledge that the origin server has. We propose the following solution: With each notification a server sends, it includes a value that an intermediary should use to calculate the Max-Age.

To summarize:

- o A notification doesn't have a Max-Age; it's fresh until the next notification arrives. A notification is the promise for another notification that will arrive at latest after Next-Notification-At-Latest. This value is included with every notification. The promise includes that the server attempts to transmit a notification to the client for the promised time span, even if the client does not seem to respond, e.g., due to a temporary network outage.
- o A notification also contains another value, called Max-Age-Hint. This value is used by a cache to calculate a Max-Age for the representation if needed. In a cache, the Max-Age-Hint of a representation is counted down like Max-Age. When it reaches zero, however, the representation can be still used to satisfy requests, but is non-cacheable (i.e., Max-Age is 0). The Max-Age-Hint must be less than or equal to Next-Notification-At-Latest.

We see two possible ways to encode Next-Notification-At-Latest and Max-Age-Hint in a message:

- o The first way is to require the values of Next-Notification-At-Latest and Max-Age-Hint to be the same, although they are conceptually unrelated. Then, a single option in the message can be used to hold both values.
- o The second way is to include two options, one for Next-Notification-At-Latest and one for Max-Age-Hint. Since Next-Notification-At-Latest is less than or equal to Max-Age-Hint, the first option should indicate Max-Age-Hint, and the second option Next-Notification-At-Latest minus Max-Age-Hint with a default value of 0.

## 6. CONNECT

[RFC2817] defines the HTTP CONNECT method to establish a TCP tunnel through a proxy so that end-to-end TLS connections can be made through the proxy. Recently, a requirement for similar functionality has been discussed for CoAP. This section defines a straw-man CONNECT method and related methods and response codes for CoAP.

(IANA considerations for this section TBD.)

### 6.1. Requesting a Tunnel with CONNECT

CONNECT is allocated as a new method code in the "CoAP Method Codes" registry. When a client makes a CONNECT request to an intermediary, the intermediary evaluates the Uri-Host, Uri-Port, and/or the authority part of the Proxy-Uri Options in a way that is defined by the security policy of the intermediary. If the security policy allows the allocation of a tunnel based on these parameters, the method returns an empty payload and a response code of 2.30 Tunnel Established. Other possible response codes include 4.03 Forbidden.

It may be the case that the intermediary itself can only reach the requested origin server through another intermediary. In this case, the first intermediary SHOULD make a CONNECT request of that next intermediary, requesting a tunnel to the authority. A proxy MUST NOT respond with any 2.xx status code unless it has either a direct or tunnel connection established to the authority.

An origin server which receives a CONNECT request for itself MAY respond with a 2.xx status code to indicate that a tunnel is established to itself.

Code 2.30 "Tunnel Established" is allocated as a new response code in the "CoAP Response Codes" registry.

### 6.2. Using a CONNECT Tunnel

Any successful (2.xx) response to a CONNECT request indicates that the intermediary has established a tunnel to the requested host and port. The tunnel is bound to the requesting end-point and the Token supplied in the request (as always, the default Token is admissible). The tunnel can be used by the client by making a DATAGRAM request.

DATAGRAM is allocated as a new method code in the "CoAP Method Codes" registry. When a client makes a DATAGRAM request to an intermediary, the intermediary looks up the tunnel bound to the client end-point and Token supplied in the DATAGRAM request (no other Options are permitted). If a tunnel is found and the intermediary's security

policy permits, the intermediary forwards the payload of the DATAGRAM request as the UDP payload towards the host and port established for the tunnel. No response is defined for this request (note that the request can be given as a CON or NON request; for CON, there will be an ACK on the message layer if the tunnel exists).

The security policy on the intermediary may restrict the allowable payloads based on its security policy, possibly considering host and port. An inadmissible payload SHOULD cause a 4.03 Forbidden response with a diagnostic message as payload.

The UDP payload of any datagram received from the tunnel and admitted by the security policy is forwarded to the client as the payload of a 2.31 "Datagram Received" response. The response does not carry any Option except for Token, which identifies the tunnel towards the client.

Code 2.31 "Datagram Received" is allocated as a new response code in the "CoAP Response Codes" registry.

An origin server that has established a tunnel to itself processes the CoAP payloads of related DATAGRAM requests as it would process an incoming UDP payload, and forwards what would be outgoing UDP payloads in 2.31 "Datagram Received" responses.

### 6.3. Closing down a CONNECT Tunnel

A 2.31 "Datagram Received" response may be replied to with a RST, which closes down the tunnel. Similarly, the Token used in the tunnel may be reused by the client for a different purpose, which also closes down the tunnel.

## 7. IANA Considerations

This draft adds option numbers to Table 2 of [I-D.ietf-core-coap]:

| Number | Name                | Reference |
|--------|---------------------|-----------|
| 14     | Registered-Elective | [RFCXXXX] |
| 22     | Patience            | [RFCXXXX] |
| 24     | Pledge              | [RFCXXXX] |
| 25     | Registered-Critical | [RFCXXXX] |

Table 1: New CoAP Option Numbers

This draft adds a suboption registry, initially empty.

| Number     | Name                        | Reference |
|------------|-----------------------------|-----------|
| 0..127     | (allocate on export review) | [RFCXXXX] |
| 128..16383 | (allocate fcfs)             | [RFCXXXX] |

Table 2: CoAP Suboption Numbers

## 8. Security Considerations

TBD.

## 9. Acknowledgements

This work was partially funded by the Klaus Tschira Foundation and by Intel Corporation.

Of course, much of the content of this draft is the result of discussions with the [I-D.ietf-core-coap] authors.

Patience and Leisure were influenced by a mailing list discussion with Esko Dijk, Kepeng Li, and Salvatore Loreto - thanks!

## 10. References

### 10.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-05 (work in progress), March 2012.
- [I-D.ietf-httpbis-pl-messaging]  
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part  
1: Message Routing and Syntax",  
draft-ietf-httpbis-pl-messaging-20 (work in progress),  
July 2012.
- [I-D.ietf-httpbis-p4-conditional]  
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part  
4: Conditional Requests",  
draft-ietf-httpbis-p4-conditional-20 (work in progress),  
July 2012.
- [I-D.ietf-httpbis-p6-cache]  
Fielding, R., Lafon, Y., Nottingham, M., and J. Reschke,  
"HTTP/1.1, part 6: Caching",  
draft-ietf-httpbis-p6-cache-20 (work in progress),  
July 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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, June 1999.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data  
Encodings", RFC 4648, October 2006.
- [RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric  
Values in Protocols", RFC 6256, May 2011.

### 10.2. Informative References

- [CoRE201] "Multiple Location options need to be processed as a  
unit", CoRE ticket #201, 2012,



- <<http://trac.tools.ietf.org/wg/core/trac/ticket/201>>.
- [CoRE214] "Adopt vendor-defined option into core-coap", CoRE ticket #214, 2012,  
<<http://trac.tools.ietf.org/wg/core/trac/ticket/214>>.
- [CoRE230] "Multiple Location options need to be processed as a unit", CoRE ticket #230, 2012,  
<<http://trac.tools.ietf.org/wg/core/trac/ticket/230>>.
- [CoRE241] "Proxy Safe & Cache Key indication for options", CoRE ticket #241, 2012,  
<<http://trac.tools.ietf.org/wg/core/trac/ticket/241>>.
- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000.
- [RFC1924] Elz, R., "A Compact Representation of IPv6 Addresses", RFC 1924, April 1996.
- [RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6648] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, June 2012.

## Appendix A. The Nursery (Things that still need to ripen a bit)

### A.1. Envelope Options

As of [I-D.ietf-core-coap], options can take one of four types, two of which are mostly identical:

- o uint: A non-negative integer which is represented in network byte order using a variable number of bytes (see [I-D.ietf-core-coap] Appendix A);
- o string: a sequence of bytes that is nominally a Net-Unicode string [RFC5198];
- o opaque: a sequence of bytes.
- o empty (not explicitly identified as a fourth type in [I-D.ietf-core-coap]).

It turns out some options would benefit from some internal structure. Also, it may be a good idea to be able to bundle multiple options into one, in order to ensure consistency for a set of elective options that need to be processed all or nothing (i.e., the option becomes critical as soon as another option out of the set is processed, too).

In this section, we introduce a fifth CoAP option type: Envelope options.

An envelope option is a sequence of bytes that looks and is interpreted exactly like a CoAP sequence of options. Instead of an option count or an end-of-option marker, the sequence of options is terminated by the end of the envelope option.

The nested options (options inside the envelope option) may come from the same number space as the top-level CoAP options, or the envelope option may define its own number space - this choice needs to be defined for each envelope option.

If the top-level number space is used, the envelope option typically will restrict the set of options that actually can be used in the envelope. In particular, it is unlikely that an envelope option will allow itself inside the envelope (this would be a recursive option).

Envelope options are a general, but simple mechanism. Some of its potential uses are illustrated by two examples in the cemetery: Appendix B.1 and Appendix B.2. (Each of these examples has its own merits and demerits, which led us to decide not to pursue either of

them right now, but this should be discussed separately from the concept of Envelope options employed in the examples.)

#### A.2. Payload-Length Option

Not all transport mappings may provide an unambiguous length of the CoAP message. For UDP, it may also be desirable to pack more than one CoAP message into one UDP payload (aggregation); in that case, for all but the last message there needs to be a way to delimit the payload of that message.

This can be solved using a new option, the Payload-Length option. If this option is present, the value of this option is an unsigned integer giving the length of the payload of the message (note that this integer can be zero for a zero-length payload, which can in turn be represented by a zero-length option value). (In the UDP aggregation case, what would have been in the payload of this message after "payload-length" bytes is then actually one or more additional messages.)

#### A.3. URI Authorities with Binary Addresses

One problem with the way URI authorities are represented in the URI syntax is that the authority part can be very bulky if it encodes an IPv6 address in ASCII.

Proposal: Provide an option "Uri-Authority-Binary" that can be an even number of bytes between 2 and 18 except 12 or 14.

- o If the number of bytes is 2, the destination IP address of the packet transporting the CoAP message is implied.
- o If the number of bytes is 4 or 6, the first four bytes of the option value are an IPv4 address in binary.
- o If the number of bytes is 8 or 10, the first eight bytes are the lower 64 bits of an IPv6 address; the upper eight bytes are implied from the destination address of the packet transporting the CoAP message.
- o If the number of bytes is 16 or 18, the first 16 bytes are an IPv6 address.
- o If two more bytes remain, this is a port number (as always in network byte order).

The resulting authority is (conceptually translated into ASCII and) used in place of an Uri-Authority option, or inserted into a Proxy-

Uri. Examples:

| Proxy-Uri      | Uri-Authority-Binary     | Uri-Path | URI                         |
|----------------|--------------------------|----------|-----------------------------|
| (none)         | (none)                   | (none)   | "/"                         |
| (none)         | (none)                   | 'temp'   | "/temp"                     |
| (none)         | 2 bytes: 61616           | 'temp'   | "coap://[DA]:61616/temp"    |
| (none)         | 16 bytes: 2000::1        | temp     | "coap://[2000::1]/temp"     |
| 'http://'      | 10 bytes: ::123:45 + 616 | (none)   | "http://[DA::123:45]:616"   |
| 'http:///temp' | 18 bytes: 2000::1 + 616  | (none)   | "http://[2000::1]:616/temp" |

#### A.4. Length-aware number encoding (o256)

The number encoding defined in Appendix A of [I-D.ietf-core-coap] has one significant flaw: Every number has an infinite number of representations, which can be derived by adding leading zero bytes. This runs against the principle of minimizing unnecessary choice. The resulting uncertainty in encoding ultimately leads to unnecessary interoperability failures. (It also wastes a small fraction of the encoding space, i.e., it wastes bytes.)

We could solve the first, but not the second, by outlawing leading zeroes, but then we have to cope with error cases caused by illegal values, another source of interoperability problems.

The number encoding "o256" defined in this section avoids this flaw. The suggestion is not to replace CoAP's "uint" encoding wholesale (CoAP is already too widely implemented for such a change), but to consider this format for new options.

The basic requirements for such an encoding are:

- o numbers are encoded as a sequence of zero or more bytes
- o each number has exactly one encoding

- o for  $a < b$ ,  $\text{encoding-size}(a) \leq \text{encoding-size}(b)$  -- i.e., with larger numbers, the encoding only gets larger, never smaller again.
- o within each encoding size (0 bytes, 1 byte, etc.), lexicographical ordering of the bytes is the same as numeric ordering

Obviously, there is only one encoding that satisfies all these requirements. As illustrated by Figure 6, this is unambiguously derived by

1. enumerating all possible byte sequences, ordered by length and within the same length in lexicographic ordering, and,
2. assigning sequential cardinals.

```

0x'' -> 0
0x'00' -> 1
0x'01' -> 2
0x'02' -> 3
...
0x'fe' -> 255
0x'ff' -> 256
0x'0000' -> 257
0x'0001' -> 258
...
0x'fefd' -> 65534
0x'fefe' -> 65535
0x'feff' -> 65536
...
0x'ffff' -> 65792
0x'000000' -> 65793
0x'000001' -> 65794

```

Figure 6: Enumerating byte sequences by length and then lexicographic order

This results in an exceedingly simple algorithm: each byte is interpreted in the base-256 place-value system, but stands for a number between 1 and 256 instead of 0 to 255. We therefore call this encoding "o256" (one-to-256). 0 is always encoded in zero bytes; 1 to 256 is one byte, 257 (0x101) to 65792 (0x10100) is two bytes, 65793 (0x10101) to 16843008 (0x1010100) is three bytes, etc.

To further illustrate the algorithmic simplicity, pseudocode for encoding and decoding is given in Figure 7 and Figure 8, respectively (in the encoder, "prepend" stands for adding a byte at the `_leading_`

edge, the requirement for which is a result of the network byte order). Note that this differs only in a single subtraction/addition (resp.) of one from the canonical algorithm for Appendix A uints.

```
while num > 0
  num -= 1
  prepend(num & 0xFF)
  num >>= 8
end
```

Figure 7: o256 encoder (pseudocode)

```
num = 0
each_byte do |b|
  num <<= 8
  num += b + 1
end
```

Figure 8: o256 decoder (pseudocode)

On a more philosophical note, it can be observed that o256 solves the inverse problem of Self-Delimiting Numeric Values (SDNV) [RFC6256]: SDNV encodes variable-length numbers together with their length (allowing decoding without knowing their length in advance, deriving delimiting information from the number encoding). o256 encodes variable-length numbers when there is a way to separately convey the length (as in CoAP options), encoding (and later deriving) a small part of the numeric value into/from that size information.

#### A.5. SMS encoding

For use in SMS applications, CoAP messages can be transferred using SMS binary mode. However, there is operational experience showing that some environments cannot successfully send a binary mode SMS.

For transferring SMS in character mode (7-bit characters), base64-encoding [RFC4648] is an obvious choice. 3 bytes of message (24 bits) turn into 4 characters, which consume 28 bits. The overall overhead is approximately 17 %; the maximum message size is 120 bytes (160 SMS characters).

If a more compact encoding is desired, base85 encoding can be employed (however, probably not the version defined in [RFC1924] -- instead, the version used in tools such as btoa and PDF should be chosen). However, this requires division operations. Also, the base85 character set includes several characters that cannot be transferred in a single 7-bit unit in SMS and/or are known to cause

operational problems. A modified base85 character set can be defined to solve the latter problem. 4 bytes of message (32 bits) turn into 5 characters, which consume 35 bits. The overall overhead is approximately 9.3 %; the resulting maximum message size is 128 bytes (160 SMS characters).

Base64 and base85 do not make use of the fact that much CoAP data will be ASCII-based. Therefore, we define the following experimental SMS encoding.

#### A.5.1. ASCII-optimized SMS encoding

Not all 128 theoretically possible SMS characters are operationally free of problems. We therefore define:

Shunned code characters: @ sign, as it maps to 0x00

LF and CR signs (0x0A, 0x0D)

uppercase C cedilla (0x09), as it is often mistranslated in gateways

ESC (0x1B), as it is used in certain character combinations only

Some ASCII characters cannot be transferred in the base SMS character set, as their code positions are taken by non-ASCII characters. These are simply encoded with their ASCII code positions, e.g., an underscore becomes a section mark (even though underscore has a different code position in the SMS character set).

Equivalently translated input bytes: \$, @, [, \, ], ^, \_, ` , {, |, }, ~, DEL

In other words, bytes 0x20 to 0x7F are encoded into the same code positions in the 7-bit character set.

Out of the remaining code characters, the following SMS characters are available for encoding:

Non-equivalently translated (NET) code characters: 0x01 to 0x08, (8 characters)

0x0B, 0x0C, (2 characters)

0x0E to 0x1A, (13 characters)

0x1C to 0x1F, (4 characters)

Of the 27 NET code characters, 18 are taken as prefix characters (see below), and 8 are defined as directly translated characters:

Directly translated bytes: Equivalently translated input bytes are represented as themselves

0x00 to 0x07 are represented as 0x01 to 0x08

This leaves 0x08 to 0x1F and 0x80 to 0xFF. Of these, the bytes 0x80 to 0x87 and 0xA0 to 0xFF are represented as the bytes 0x00 to 0x07 (represented by characters 0x01 to 0x08) and 0x20 to 0x7F, with a prefix of 1 (see below). The characters 0x08 to 0x1F are represented as the characters 0x28 to 0x3F with a prefix of 2 (see below). The characters 0x88 to 0x9F are represented as the characters 0x48 to 0x5F with a prefix of 2 (see below). (Characters 0x01 to 0x08, 0x20 to 0x27, 0x40 to 0x47, and 0x60 to 0x7f with a prefix of 2 are reserved for future extensions, which could be used for some backreferencing or run-length compression.)

Bytes that do not need a prefix (directly translated bytes) are sent as is. Any byte that does need a prefix (i.e., 1 or 2) is preceded by a prefix character, which provides a prefix for this and the following two bytes as follows:

|      |     |  |      |     |
|------|-----|--|------|-----|
| 0x0B | 100 |  | 0x15 | 200 |
| 0x0C | 101 |  | 0x16 | 201 |
| 0x0E | 102 |  | 0x17 | 202 |
| 0x0F | 110 |  | 0x18 | 210 |
| 0x10 | 111 |  | 0x19 | 211 |
| 0x11 | 112 |  | 0x1A | 212 |
| 0x12 | 120 |  | 0x1C | 220 |
| 0x13 | 121 |  | 0x1D | 221 |
| 0x14 | 122 |  | 0x1E | 222 |

(This leaves one non-shunned character, 0x1F, for future extension.)



The coding overhead of this encoding for random bytes is similar to Base85, without the need for a division/multiplication. For bytes that are mostly ASCII characters, the overhead can easily become negative. (Conversely, for bytes that are more likely to be non-ASCII than in a random sequence of bytes, the overhead becomes greater.)

So, for instance, for the CoAP message in Figure 9:

|              |     |         |       |  |
|--------------|-----|---------|-------|--|
| ver          | tt  | code    | mid   |  |
| 1            | ack | 2.05    | 17033 |  |
| content_type |     | 40      |       |  |
| token        |     | sometok |       |  |

```

3c 2f 3e 3b 74 69 74 6c 65 3d 22 47 65 6e 65 72 |</>;title="Gener
61 6c 20 49 6e 66 6f 22 3b 63 74 3d 30 2c 3c 2f |al Info";ct=0,</
74 69 6d 65 3e 3b 69 66 3d 22 63 6c 6f 63 6b 22 |time>;if="clock"
3b 72 74 3d 22 54 69 63 6b 73 22 3b 74 69 74 6c |;rt="Ticks";titl
65 3d 22 49 6e 74 65 72 6e 61 6c 20 43 6c 6f 63 |e="Internal Cloc
6b 22 3b 63 74 3d 30 2c 3c 2f 61 73 79 6e 63 3e |k";ct=0,</async>
3b 63 74 3d 30 |;ct=0

```

Figure 9: CoAP response message as captured and decoded

The 116 byte unencoded message is shown as ASCII characters in Figure 10 (\xDD stands for the byte with the hex digits DD):

```

bEB\x89\x11(\xA7sometok</>;title="General Info";ct=0,</time>
;if="clock";rt="Ticks";title="Internal Clock";ct=0,</async>;ct=0

```

Figure 10: CoAP response message shown as unencoded characters

The equivalent SMS encoding is shown as equivalent-coded SMS characters in Figure 11 (7 bits per character, \x12 is a 220 prefix and \x0B is a 100 prefix, the rest is shown in equivalent encoding), adding two characters of prefix overhead, for a total length of 118 7-bit characters or 104 (103.25 plus padding) bytes:

```

bEB\x12I1(\x0B'sometok</>;title="General Info";ct=0,</time>
;if="clock";rt="Ticks";title="Internal Clock";ct=0,</async>;ct=0

```

Figure 11: CoAP response message shown as SMS-encoded characters

## Appendix B. The Cemetery (Things we won't do)

This annex documents roads that the WG decided not to take, in order to spare readers from reinventing them in vain.

### B.1. Example envelope option: solving #230

Ticket #230 [CoRE230] points out a design flaw of [I-D.ietf-core-coap]: When we split the elective Location option of draft -01 into multiple elective options, we made it possible that an implementation might process some of these and ignore others, leading to an incorrect interpretation of the Location expressed by the server.

There are several more or less savory solutions to #230.

Each of the elective options that together make up the Location could be defined in such a way that it makes a requirement on the processing of the related option (essentially revoking their elective status once the option under consideration is actually processed). This falls flat as soon as another option is defined that would also become part of the Location: existing implementations would not know that the new option is also part of the cluster that is re-interpreted as critical. The potential future addition of Location-Host and Location-Port makes this a valid consideration.

A better solution would be to define an elective Envelope Option called Location. Within a Location Option, the following top-level options might be allowed (now or in the future):

- o Uri-Host
- o Uri-Port
- o Uri-Path
- o Uri-Query

This would unify the code for interpreting the top-level request options that indicate the request URI with the code that interprets the Location URI.

The four options listed are all critical, while the envelope is elective. This gives exactly the desired semantics: If the envelope is processed at all (which is elective), the nested options are critical and all need to be processed.

## B.2. Example envelope option: proxy-elective options

Another potential application of envelope options is motivated by the observation that new critical options might not be implemented by all proxies on the CoAP path to an origin server. So that this does not become an obstacle to introducing new critical options that are of interest only to client and origin server, the client might want to mark some critical options proxy-elective, i.e. elective for a proxy but still critical for the origin server.

One way to do this would be an Envelope option, the Proxy-Elective Option. A client might bundle a number of critical options into a critical Proxy-Elective Option. A proxy that processes the message is obliged to process the envelope (or reject the message), where processing means passing on the nested options towards the origin server (preferably again within a Proxy-Elective option). It can pass on the nested options, even ones unknown to the proxy, knowing that the client is happy with proxies not processing all of them.

(The assumption here is that the Proxy-Elective option becomes part of the base standard, so all but the most basic proxies would know how to handle it.)

## B.3. Stateful URI compression

Is the approximately 25 % average saving achievable with Huffman-based URI compression schemes worth the complexity? Probably not, because much higher average savings can be achieved by introducing state.

Henning Schulzrinne has proposed for a server to be able to supply a shortened URI once a resource has been requested using the full-length URI. Let's call such a shortened referent a `_Temporary Resource Identifier_`, `_TeRI_` for short. This could be expressed by a response option as shown in Figure 12.

```

0
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+---+---+---+---+---+---+---+---+---+
|   duration   |   TeRI...
+---+---+---+---+---+---+---+---+

```

Figure 12: Option for offering a TeRI in a response

The TeRI offer option indicates that the server promises to offer this resources under the TeRI given for at least the time given as the duration. Another TeRI offer can be made later to extend the duration.

Once a TeRI for a URI is known (and still within its lifetime), the client can supply a TeRI instead of a URI in its requests. The same option format as an offer could be used to allow the client to indicate how long it believes the TeRI will still be valid (so that the server can decide when to update the lifetime duration). TeRIs in requests could be distinguished from URIs e.g. by using a different option number.

Proposal: Add a TeRI option that can be used in CoAP requests and responses.

Add a way to indicate a TeRI and its duration in a link-value.

Do not add any form of stateless URI encoding.

Benefits: Much higher reduction of message size than any stateless URI encoding could achieve.

As the use of TeRIs is entirely optional, minimal complexity nodes can get by without implementing them.

Drawbacks: Adds considerable state and complexity to the protocol.

It turns out that real CoAP URIs are short enough that TeRIs are not needed.

(Discuss the security implications of TeRIs.)

#### B.4. Beyond 270 bytes in a single option

The authors would argue that 270 as the maximum length of an option is already beyond the "painless" threshold.

If that is not the consensus of the WG, the scheme can easily be extended as in Figure 13:

[illegible]

Figure 13: Ridiculously Long Option Header

The infinite number of obvious variations on this scheme are left as an exercise to the reader.

Again, as a precaution to future extensions, the current encoding for length 270 (eight ones in the extension byte) could be marked as reserved today.

### B.5. Beyond 15 options

(This section keeps discussion that is no longer needed as we have agreed to do what is documented in Appendix B.6).

The limit of 15 options is motivated by the fixed four-bit field "OC" that is used for indicating the number of options in the fixed-length CoAP header (Figure 14).

[illegible]

Figure 14: Four-byte fixed header in a CoAP Message

Note that there is another fixed four-bit field in CoAP: the option length (Figure 15 - note that this figure is not to the same scale as

the previous figure):

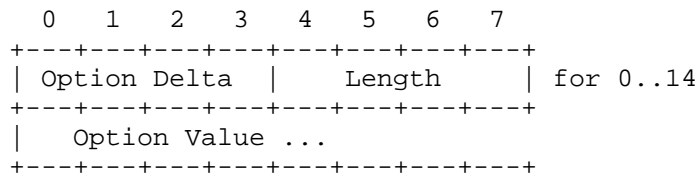


Figure 15: Short Option Header

Since 15 is unacceptable for a maximum option length, the all-ones value (15) was taken out of the set of allowable values for the short header, and a long header was introduced that allows the insertion of an extension byte (Figure 16):

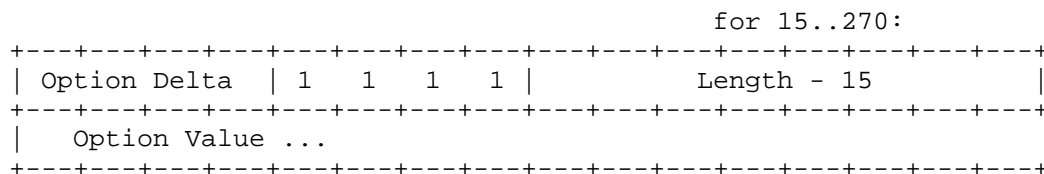


Figure 16: Long Option Header

We might want to use the same technique for the CoAP header as well. There are two obvious places where the extension byte could be placed:

1. right after the byte carrying the OC field, so the structure is the same as for the option header;
2. right after the fixed-size CoAP header.

Both solutions lose the fixed-size-ness of the CoAP header.

Solution 1 has the disadvantage that the CoAP header is also changing in structure: The extension byte is wedged between the first and the second byte of the CoAP header. This is unfortunate, as the number of options only comes into play when the option processing begins, so it is more natural to use solution 2 (Figure 17):

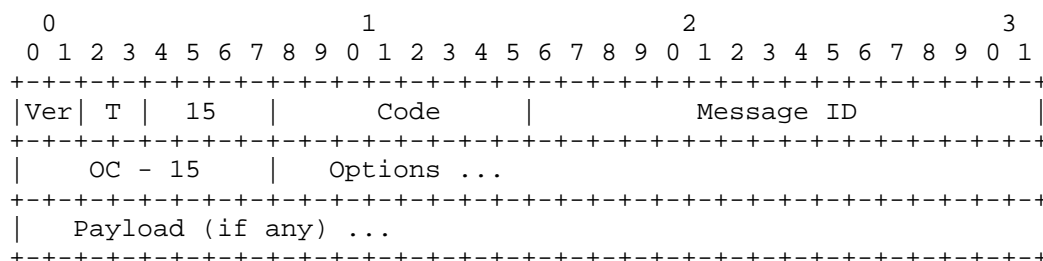


Figure 17: Extended header for CoAP Messages with 15+ options

This would allow for up to 270 options in a CoAP message, which is very likely way beyond the "painless" threshold.

#### B.5.1.1. Implementation considerations

For a message decoder, this extension creates relatively little pain, as the number of options only becomes interesting when the encoding turns to the options part of the message, which is then simply lead in by the extension byte if the four-bit field is 15.

For a message encoder, this extension is not so rosy. If the encoder is constructing the message serially, it may not know in advance whether the number of options will exceed 14. None of the following implementation strategies is particularly savory, but all of them do work:

1. Encode the options serially under the assumption that the number of options will be 14 or less. When the 15th option needs to be encoded, abort the option encoding, and restart it from scratch one byte further to the left.
2. Similar to 1, except that the bytes already encoded are all moved one byte to right, the extension byte is inserted, and the option encoding process is continued.
3. The encoder always leaves space for the extension byte (at least if it can't prove the number will be less than 14). If the extension byte is not needed, an Option 0 with length 0 is encoded instead (i.e., one byte is wasted - this option is elective and will be ignored by the receiver).

As a minimum, to enable strategy 3, the option 0 should be reserved at least for the case of length=0.

#### B.5.2. What should we do now?

As a minimum proposal for the next version of CoAP, the value 15 for OC should be marked as reserved today.

#### B.5.3. Alternatives

One alternative that has been discussed previously is to have an "Options" Option, which allows the carriage of multiple options in the belly of a single one. This could also be used to carry more than 15 options. However:

- o The conditional introduction of an Options option has implementation considerations that are likely to be more severe than the ones listed above;
- o since 270 bytes may not be enough for the encoding of `_all_` options, the "Options" option would need to be repeatable. This creates many different ways to encode the same message, leading to combinatorial explosion in test cases for ensuring interoperability.

#### B.5.4. Alternative: Going to a delimiter model

Another alternative is to spend the additional byte not as an extended count, but as an option terminator.

#### B.6. Implementing the option delimiter for 15 or more options

Implementation note: As can be seen from the proof of concept code in Figure 18, the actual implementation cost for a decoder is around 4 lines of code (or about 8-10 machine code instructions).

```
while numopt > 0
  nextbyte = ... get next byte

  if numopt == 15                # new
    break if nextbyte == 0xF0    # new
  else                           # new
    numopt -= 1
  end                             # new

  ... decode the delta and length from nextbyte and handle them
end
```

Figure 18: Implementing the Option Terminator



Similarly, creating the option terminator needs about four more lines (not marked "old" in the C code in Figure 19).

```
b0 = 0x40 + (tt << 4);          /* old */
buffer[0] = b0 + 15;            /* guess first byte */

.... encode options ....      /* old */

if (option_count >= 15 || first_fragment_already_shipped)
    buffer[pos++] = 0xF0;       /* use delimiter */
else
    buffer[0] = b0 + option_count; /* old: backpatch */
```

Figure 19: Creating the Option Terminator

## Appendix C. Experimental Options

This annex documents proposals that need significant additional discussion before they can become part of (or go back to) the main CoAP specification. They are not dead, but might die if there turns out to be no good way to solve the problem.

### C.1. Options indicating absolute time

HTTP has a number of headers that may indicate absolute time:

- o "Date", defined in Section 14.18 in [RFC2616] (Section 9.3 in [I-D.ietf-httpbis-pl-messaging]), giving the absolute time a response was generated;
- o "Last-Modified", defined in Section 14.29 in [RFC2616], (Section 6.6 in [I-D.ietf-httpbis-p4-conditional], giving the absolute time of when the origin server believes the resource representation was last modified;
- o "If-Modified-Since", defined in Section 14.25 in [RFC2616], "If-Unmodified-Since", defined in Section 14.28 in [RFC2616], and "If-Range", defined in Section 14.27 in [RFC2616] can be used to supply absolute time to gate a conditional request;
- o "Expires", defined in Section 14.21 in [RFC2616] (Section 3.3 in [I-D.ietf-httpbis-p6-cache]), giving the absolute time after which a response is considered stale.
- o The more obscure headers "Retry-After", defined in Section 14.37 in [RFC2616], and "Warning", defined in section 14.46 in [RFC2616], also may employ absolute time.

[I-D.ietf-core-coap] defines a single "Date" option, which however "indicates the creation time and date of a given resource representation", i.e., is closer to a "Last-Modified" HTTP header. HTTP's caching rules [I-D.ietf-httpbis-p6-cache] make use of both "Date" and "Last-Modified", combined with "Expires". The specific semantics required for CoAP needs further consideration.

In addition to the definition of the semantics, an encoding for absolute times needs to be specified.

In UNIX-related systems, it is customary to indicate absolute time as an integer number of seconds, after midnight UTC, January 1, 1970. Unless negative numbers are employed, this time format cannot represent time values prior to January 1, 1970, which probably is not required for the uses of absolute time in CoAP.

If a 32-bit integer is used and allowance is made for a sign-bit in a local implementation, the latest UTC time value that can be represented by the resulting 31 bit integer value is 03:14:07 on January 19, 2038. If the 32-bit integer is used as an unsigned value, the last date is 2106-02-07, 06:28:15.

The reach can be extended by: - moving the epoch forward, e.g. by 40 years (= 1262304000 seconds) to 2010-01-01. This makes it impossible to represent Last-Modified times in that past (such as could be gatewayed in from HTTP). - extending the number of bits, e.g. by one more byte, either always or as one of two formats, keeping the 32-bit variant as well.

Also, the resolution can be extended by expressing time in milliseconds etc., requiring even more bits (e.g., a 48-bit unsigned integer of milliseconds would last well after year 9999.)

For experiments, an experimental "Date" option is defined with the semantics of HTTP's "Last-Modified". It can carry an unsigned integer of 32, 40, or 48 bits; 32- and 40-bit integers indicate the absolute time in seconds since 1970-01-01 00:00 UTC, while 48-bit integers indicate the absolute time in milliseconds since 1970-01-01 00:00 UTC.

However, that option is not really that useful until there is a "If-Modified-Since" option as well.

(Also: Discuss nodes without clocks.)

## C.2. Representing Durations

Various message types used in CoAP need the representation of \*durations\*, i.e. of the length of a timespan. In SI units, these are measured in seconds. CoAP durations represent integer numbers of seconds, but instead of representing these numbers as integers, a more compact single-byte pseudo-floating-point (pseudo-FP) representation is used (Figure 20).

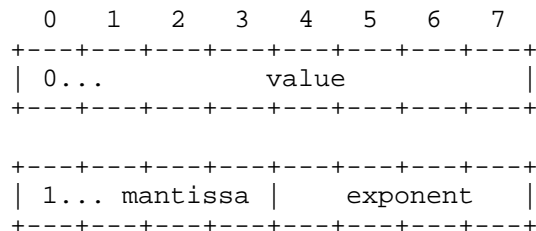


Figure 20: Duration in (8,4) pseudo-FP representation

If the high bit is clear, the entire n-bit value (including the high bit) is the decoded value. If the high bit is set, the mantissa (including the high bit, with the exponent field cleared out but still present) is shifted left by the exponent to yield the decoded value.

The (n,e)-pseudo-FP format can be decoded with a single line of code (plus a couple of constant definitions), as demonstrated in Figure 21.

```
#define N 8
#define E 4
#define HIBIT (1 << (N - 1))
#define EMASK ((1 << E) - 1)
#define MMASK ((1 << N) - 1 - EMASK)

#define DECODE_8_4(r) (r < HIBIT ? r : (r & MMASK) << (r & EMASK))
```

Figure 21: Decoding an (8,4) pseudo-FP value

Note that a pseudo-FP encoder needs to consider rounding; different applications of durations may favor rounding up or rounding down the value encoded in the message.

The highest pseudo-FP value, represented by an all-ones byte (0xFF), is reserved to indicate an indefinite duration. The next lower value (0xEF) is thus the highest representable value and is decoded as 7340032 seconds, a little more than 12 weeks.

### C.3. Rationale

Where CPU power and memory is abundant, a duration can almost always be adequately represented by a non-negative floating-point number representing that number of seconds. Historically, many APIs have also used an integer representation, which limits both the resolution (e.g., if the integer represents the duration in seconds) and often the range (integer machine types have range limits that may become relevant). UNIX's "time\_t" (which is used for both absolute time and durations) originally was a signed 32-bit value of seconds, but was later complemented by an additional integer to add microsecond ("struct timeval") and then later nanosecond ("struct timespec") resolution.

Three decisions need to be made for each application of the concept of duration:

- o the *\*resolution\**. What rounding error is acceptable?
- o the *\*range\**. What is the maximum duration that needs to be represented?
- o the *\*number of bits\** that can be expended.

Obviously, these decisions are interrelated. Typically, a large range needs a large number of bits, unless resolution is traded. For most applications, the actual requirement for resolution are limited for longer durations, but can be more acute for shorter durations.

#### C.4. Pseudo-Floating Point

Constrained systems typically avoid the use of floating-point (FP) values, as

- o simple CPUs often don't have support for floating-point datatypes
- o software floating-point libraries are expensive in code size and slow.

In addition, floating-point datatypes used to be a significant element of market differentiation in CPU design; it has taken the industry a long time to agree on a standard floating point representation.

These issues have led to protocols that try to constrain themselves to integer representation even where the ability of a floating point representation to trade range for resolution would be beneficial.

The idea of introducing *\_pseudo-FP\_* is to obtain the increased range provided by embedding an exponent, without necessarily getting stuck with hardware datatypes or inefficient software floating-point libraries.

For the purposes of this draft, we define an (n,e)-pseudo-FP as a fixed-length value of n bits, e of which may be used for an exponent. Figure 20 illustrates an (8,4)-pseudo-FP value.

If the high bit is clear, the entire n-bit value (including the high bit) is the decoded value. If the high bit is set, the mantissa (including the high bit, but with the exponent field cleared out) is shifted left by the exponent to yield the decoded value.

The (n,e)-pseudo-FP format can be decoded with a single line of code (plus a couple of constant definition), as demonstrated in Figure 21.

Only non-negative numbers can be represented by this format. It is designed to provide full integer resolution for values from 0 to  $2^{(n-1)}-1$ , i.e., 0 to 127 in the (8,4) case, and a mantissa of  $n-e$  bits from  $2^{(n-1)}$  to  $(2^n-2^e)*2^{(2^e-1)}$ , i.e., 128 to 7864320 in the (8,4) case. By choosing  $e$  carefully, resolution can be traded against range.

Note that a pseudo-FP encoder needs to consider rounding; different applications of durations may favor rounding up or rounding down the value encoded in the message. This requires a little more than a single line of code (which is left as an exercise to the reader, as the most efficient expression depends on hardware details).

### C.5. A Duration Type for CoAP

CoAP needs durations in a number of places. In [I-D.ietf-core-coap], durations occur in the option "Subscription-lifetime" as well as in the option "Max-age". (Note that the option "Date" is not a duration, but a point in time.) Other durations of this kind may be added later.

Most durations relevant to CoAP are best expressed with a minimum resolution of one second. More detailed resolutions are unlikely to provide much benefit.

The range of lifetimes and caching ages are probably best kept below the order of magnitude of months. An (8,4)-pseudo-FP has the maximum value of 7864320, which is about 91 days; this appears to be adequate for a subscription lifetime and probably even for a maximum cache age. Figure 22 shows the values that can be expressed. (If a larger range for the latter is indeed desired, an (8,5)-pseudo-FP could be used; this would last 15 milleniums, at the cost of having only 3 bits of accuracy for values larger than 127 seconds.)

Proposal: A single duration type is used throughout CoAP, based on an (8,4)-pseudo-FP giving a duration in seconds.

Benefits: Implementations can use a single piece of code for managing all CoAP-related durations.

In addition, length information never needs to be managed for durations that are embedded in other data structures: All durations are expressed by a single byte.

It might be worthwhile to reserve one duration value, e.g. 0xFF, for an indefinite duration.

| Duration | Seconds | Encoded |
|----------|---------|---------|
|----------|---------|---------|

| -----    | -----      | ----- |
|----------|------------|-------|
| 00:00:00 | 0x00000000 | 0x00  |
| 00:00:01 | 0x00000001 | 0x01  |
| 00:00:02 | 0x00000002 | 0x02  |
| 00:00:03 | 0x00000003 | 0x03  |
| 00:00:04 | 0x00000004 | 0x04  |
| 00:00:05 | 0x00000005 | 0x05  |
| 00:00:06 | 0x00000006 | 0x06  |
| 00:00:07 | 0x00000007 | 0x07  |
| 00:00:08 | 0x00000008 | 0x08  |
| 00:00:09 | 0x00000009 | 0x09  |
| 00:00:10 | 0x0000000a | 0x0a  |
| 00:00:11 | 0x0000000b | 0x0b  |
| 00:00:12 | 0x0000000c | 0x0c  |
| 00:00:13 | 0x0000000d | 0x0d  |
| 00:00:14 | 0x0000000e | 0x0e  |
| 00:00:15 | 0x0000000f | 0x0f  |
| 00:00:16 | 0x00000010 | 0x10  |
| 00:00:17 | 0x00000011 | 0x11  |
| 00:00:18 | 0x00000012 | 0x12  |
| 00:00:19 | 0x00000013 | 0x13  |
| 00:00:20 | 0x00000014 | 0x14  |
| 00:00:21 | 0x00000015 | 0x15  |
| 00:00:22 | 0x00000016 | 0x16  |
| 00:00:23 | 0x00000017 | 0x17  |
| 00:00:24 | 0x00000018 | 0x18  |
| 00:00:25 | 0x00000019 | 0x19  |
| 00:00:26 | 0x0000001a | 0x1a  |
| 00:00:27 | 0x0000001b | 0x1b  |
| 00:00:28 | 0x0000001c | 0x1c  |
| 00:00:29 | 0x0000001d | 0x1d  |
| 00:00:30 | 0x0000001e | 0x1e  |
| 00:00:31 | 0x0000001f | 0x1f  |
| 00:00:32 | 0x00000020 | 0x20  |
| 00:00:33 | 0x00000021 | 0x21  |
| 00:00:34 | 0x00000022 | 0x22  |
| 00:00:35 | 0x00000023 | 0x23  |
| 00:00:36 | 0x00000024 | 0x24  |
| 00:00:37 | 0x00000025 | 0x25  |
| 00:00:38 | 0x00000026 | 0x26  |
| 00:00:39 | 0x00000027 | 0x27  |
| 00:00:40 | 0x00000028 | 0x28  |
| 00:00:41 | 0x00000029 | 0x29  |
| 00:00:42 | 0x0000002a | 0x2a  |
| 00:00:43 | 0x0000002b | 0x2b  |
| 00:00:44 | 0x0000002c | 0x2c  |
| 00:00:45 | 0x0000002d | 0x2d  |
| 00:00:46 | 0x0000002e | 0x2e  |

|          |            |      |
|----------|------------|------|
| 00:00:47 | 0x0000002f | 0x2f |
| 00:00:48 | 0x00000030 | 0x30 |
| 00:00:49 | 0x00000031 | 0x31 |
| 00:00:50 | 0x00000032 | 0x32 |
| 00:00:51 | 0x00000033 | 0x33 |
| 00:00:52 | 0x00000034 | 0x34 |
| 00:00:53 | 0x00000035 | 0x35 |
| 00:00:54 | 0x00000036 | 0x36 |
| 00:00:55 | 0x00000037 | 0x37 |
| 00:00:56 | 0x00000038 | 0x38 |
| 00:00:57 | 0x00000039 | 0x39 |
| 00:00:58 | 0x0000003a | 0x3a |
| 00:00:59 | 0x0000003b | 0x3b |
| 00:01:00 | 0x0000003c | 0x3c |
| 00:01:01 | 0x0000003d | 0x3d |
| 00:01:02 | 0x0000003e | 0x3e |
| 00:01:03 | 0x0000003f | 0x3f |
| 00:01:04 | 0x00000040 | 0x40 |
| 00:01:05 | 0x00000041 | 0x41 |
| 00:01:06 | 0x00000042 | 0x42 |
| 00:01:07 | 0x00000043 | 0x43 |
| 00:01:08 | 0x00000044 | 0x44 |
| 00:01:09 | 0x00000045 | 0x45 |
| 00:01:10 | 0x00000046 | 0x46 |
| 00:01:11 | 0x00000047 | 0x47 |
| 00:01:12 | 0x00000048 | 0x48 |
| 00:01:13 | 0x00000049 | 0x49 |
| 00:01:14 | 0x0000004a | 0x4a |
| 00:01:15 | 0x0000004b | 0x4b |
| 00:01:16 | 0x0000004c | 0x4c |
| 00:01:17 | 0x0000004d | 0x4d |
| 00:01:18 | 0x0000004e | 0x4e |
| 00:01:19 | 0x0000004f | 0x4f |
| 00:01:20 | 0x00000050 | 0x50 |
| 00:01:21 | 0x00000051 | 0x51 |
| 00:01:22 | 0x00000052 | 0x52 |
| 00:01:23 | 0x00000053 | 0x53 |
| 00:01:24 | 0x00000054 | 0x54 |
| 00:01:25 | 0x00000055 | 0x55 |
| 00:01:26 | 0x00000056 | 0x56 |
| 00:01:27 | 0x00000057 | 0x57 |
| 00:01:28 | 0x00000058 | 0x58 |
| 00:01:29 | 0x00000059 | 0x59 |
| 00:01:30 | 0x0000005a | 0x5a |
| 00:01:31 | 0x0000005b | 0x5b |
| 00:01:32 | 0x0000005c | 0x5c |
| 00:01:33 | 0x0000005d | 0x5d |
| 00:01:34 | 0x0000005e | 0x5e |



|          |            |      |
|----------|------------|------|
| 00:01:35 | 0x0000005f | 0x5f |
| 00:01:36 | 0x00000060 | 0x60 |
| 00:01:37 | 0x00000061 | 0x61 |
| 00:01:38 | 0x00000062 | 0x62 |
| 00:01:39 | 0x00000063 | 0x63 |
| 00:01:40 | 0x00000064 | 0x64 |
| 00:01:41 | 0x00000065 | 0x65 |
| 00:01:42 | 0x00000066 | 0x66 |
| 00:01:43 | 0x00000067 | 0x67 |
| 00:01:44 | 0x00000068 | 0x68 |
| 00:01:45 | 0x00000069 | 0x69 |
| 00:01:46 | 0x0000006a | 0x6a |
| 00:01:47 | 0x0000006b | 0x6b |
| 00:01:48 | 0x0000006c | 0x6c |
| 00:01:49 | 0x0000006d | 0x6d |
| 00:01:50 | 0x0000006e | 0x6e |
| 00:01:51 | 0x0000006f | 0x6f |
| 00:01:52 | 0x00000070 | 0x70 |
| 00:01:53 | 0x00000071 | 0x71 |
| 00:01:54 | 0x00000072 | 0x72 |
| 00:01:55 | 0x00000073 | 0x73 |
| 00:01:56 | 0x00000074 | 0x74 |
| 00:01:57 | 0x00000075 | 0x75 |
| 00:01:58 | 0x00000076 | 0x76 |
| 00:01:59 | 0x00000077 | 0x77 |
| 00:02:00 | 0x00000078 | 0x78 |
| 00:02:01 | 0x00000079 | 0x79 |
| 00:02:02 | 0x0000007a | 0x7a |
| 00:02:03 | 0x0000007b | 0x7b |
| 00:02:04 | 0x0000007c | 0x7c |
| 00:02:05 | 0x0000007d | 0x7d |
| 00:02:06 | 0x0000007e | 0x7e |
| 00:02:07 | 0x0000007f | 0x7f |
| 00:02:08 | 0x00000080 | 0x80 |
| 00:02:24 | 0x00000090 | 0x90 |
| 00:02:40 | 0x000000a0 | 0xa0 |
| 00:02:56 | 0x000000b0 | 0xb0 |
| 00:03:12 | 0x000000c0 | 0xc0 |
| 00:03:28 | 0x000000d0 | 0xd0 |
| 00:03:44 | 0x000000e0 | 0xe0 |
| 00:04:00 | 0x000000f0 | 0xf0 |
| 00:04:16 | 0x00000100 | 0x81 |
| 00:04:48 | 0x00000120 | 0x91 |
| 00:05:20 | 0x00000140 | 0xa1 |
| 00:05:52 | 0x00000160 | 0xb1 |
| 00:06:24 | 0x00000180 | 0xc1 |
| 00:06:56 | 0x000001a0 | 0xd1 |
| 00:07:28 | 0x000001c0 | 0xe1 |

|          |            |      |
|----------|------------|------|
| 00:08:00 | 0x000001e0 | 0xf1 |
| 00:08:32 | 0x00000200 | 0x82 |
| 00:09:36 | 0x00000240 | 0x92 |
| 00:10:40 | 0x00000280 | 0xa2 |
| 00:11:44 | 0x000002c0 | 0xb2 |
| 00:12:48 | 0x00000300 | 0xc2 |
| 00:13:52 | 0x00000340 | 0xd2 |
| 00:14:56 | 0x00000380 | 0xe2 |
| 00:16:00 | 0x000003c0 | 0xf2 |
| 00:17:04 | 0x00000400 | 0x83 |
| 00:19:12 | 0x00000480 | 0x93 |
| 00:21:20 | 0x00000500 | 0xa3 |
| 00:23:28 | 0x00000580 | 0xb3 |
| 00:25:36 | 0x00000600 | 0xc3 |
| 00:27:44 | 0x00000680 | 0xd3 |
| 00:29:52 | 0x00000700 | 0xe3 |
| 00:32:00 | 0x00000780 | 0xf3 |
| 00:34:08 | 0x00000800 | 0x84 |
| 00:38:24 | 0x00000900 | 0x94 |
| 00:42:40 | 0x00000a00 | 0xa4 |
| 00:46:56 | 0x00000b00 | 0xb4 |
| 00:51:12 | 0x00000c00 | 0xc4 |
| 00:55:28 | 0x00000d00 | 0xd4 |
| 00:59:44 | 0x00000e00 | 0xe4 |
| 01:04:00 | 0x00000f00 | 0xf4 |
| 01:08:16 | 0x00001000 | 0x85 |
| 01:16:48 | 0x00001200 | 0x95 |
| 01:25:20 | 0x00001400 | 0xa5 |
| 01:33:52 | 0x00001600 | 0xb5 |
| 01:42:24 | 0x00001800 | 0xc5 |
| 01:50:56 | 0x00001a00 | 0xd5 |
| 01:59:28 | 0x00001c00 | 0xe5 |
| 02:08:00 | 0x00001e00 | 0xf5 |
| 02:16:32 | 0x00002000 | 0x86 |
| 02:33:36 | 0x00002400 | 0x96 |
| 02:50:40 | 0x00002800 | 0xa6 |
| 03:07:44 | 0x00002c00 | 0xb6 |
| 03:24:48 | 0x00003000 | 0xc6 |
| 03:41:52 | 0x00003400 | 0xd6 |
| 03:58:56 | 0x00003800 | 0xe6 |
| 04:16:00 | 0x00003c00 | 0xf6 |
| 04:33:04 | 0x00004000 | 0x87 |
| 05:07:12 | 0x00004800 | 0x97 |
| 05:41:20 | 0x00005000 | 0xa7 |
| 06:15:28 | 0x00005800 | 0xb7 |
| 06:49:36 | 0x00006000 | 0xc7 |
| 07:23:44 | 0x00006800 | 0xd7 |
| 07:57:52 | 0x00007000 | 0xe7 |

|     |          |            |      |
|-----|----------|------------|------|
|     | 08:32:00 | 0x00007800 | 0xf7 |
|     | 09:06:08 | 0x00008000 | 0x88 |
|     | 10:14:24 | 0x00009000 | 0x98 |
|     | 11:22:40 | 0x0000a000 | 0xa8 |
|     | 12:30:56 | 0x0000b000 | 0xb8 |
|     | 13:39:12 | 0x0000c000 | 0xc8 |
|     | 14:47:28 | 0x0000d000 | 0xd8 |
|     | 15:55:44 | 0x0000e000 | 0xe8 |
|     | 17:04:00 | 0x0000f000 | 0xf8 |
|     | 18:12:16 | 0x00010000 | 0x89 |
|     | 20:28:48 | 0x00012000 | 0x99 |
|     | 22:45:20 | 0x00014000 | 0xa9 |
| 1d  | 01:01:52 | 0x00016000 | 0xb9 |
| 1d  | 03:18:24 | 0x00018000 | 0xc9 |
| 1d  | 05:34:56 | 0x0001a000 | 0xd9 |
| 1d  | 07:51:28 | 0x0001c000 | 0xe9 |
| 1d  | 10:08:00 | 0x0001e000 | 0xf9 |
| 1d  | 12:24:32 | 0x00020000 | 0x8a |
| 1d  | 16:57:36 | 0x00024000 | 0x9a |
| 1d  | 21:30:40 | 0x00028000 | 0xaa |
| 2d  | 02:03:44 | 0x0002c000 | 0xba |
| 2d  | 06:36:48 | 0x00030000 | 0xca |
| 2d  | 11:09:52 | 0x00034000 | 0xda |
| 2d  | 15:42:56 | 0x00038000 | 0xea |
| 2d  | 20:16:00 | 0x0003c000 | 0xfa |
| 3d  | 00:49:04 | 0x00040000 | 0x8b |
| 3d  | 09:55:12 | 0x00048000 | 0x9b |
| 3d  | 19:01:20 | 0x00050000 | 0xab |
| 4d  | 04:07:28 | 0x00058000 | 0xbb |
| 4d  | 13:13:36 | 0x00060000 | 0xcb |
| 4d  | 22:19:44 | 0x00068000 | 0xdb |
| 5d  | 07:25:52 | 0x00070000 | 0xeb |
| 5d  | 16:32:00 | 0x00078000 | 0xfb |
| 6d  | 01:38:08 | 0x00080000 | 0x8c |
| 6d  | 19:50:24 | 0x00090000 | 0x9c |
| 7d  | 14:02:40 | 0x000a0000 | 0xac |
| 8d  | 08:14:56 | 0x000b0000 | 0xbc |
| 9d  | 02:27:12 | 0x000c0000 | 0xcc |
| 9d  | 20:39:28 | 0x000d0000 | 0xdc |
| 10d | 14:51:44 | 0x000e0000 | 0xec |
| 11d | 09:04:00 | 0x000f0000 | 0xfc |
| 12d | 03:16:16 | 0x00100000 | 0x8d |
| 13d | 15:40:48 | 0x00120000 | 0x9d |
| 15d | 04:05:20 | 0x00140000 | 0xad |
| 16d | 16:29:52 | 0x00160000 | 0xbd |
| 18d | 04:54:24 | 0x00180000 | 0xcd |
| 19d | 17:18:56 | 0x001a0000 | 0xdd |
| 21d | 05:43:28 | 0x001c0000 | 0xed |

|     |          |            |                 |
|-----|----------|------------|-----------------|
| 22d | 18:08:00 | 0x001e0000 | 0xfd            |
| 24d | 06:32:32 | 0x00200000 | 0x8e            |
| 27d | 07:21:36 | 0x00240000 | 0x9e            |
| 30d | 08:10:40 | 0x00280000 | 0xae            |
| 33d | 08:59:44 | 0x002c0000 | 0xbe            |
| 36d | 09:48:48 | 0x00300000 | 0xce            |
| 39d | 10:37:52 | 0x00340000 | 0xde            |
| 42d | 11:26:56 | 0x00380000 | 0xee            |
| 45d | 12:16:00 | 0x003c0000 | 0xfe            |
| 48d | 13:05:04 | 0x00400000 | 0x8f            |
| 54d | 14:43:12 | 0x00480000 | 0x9f            |
| 60d | 16:21:20 | 0x00500000 | 0xaf            |
| 66d | 17:59:28 | 0x00580000 | 0xbf            |
| 72d | 19:37:36 | 0x00600000 | 0xcf            |
| 78d | 21:15:44 | 0x00680000 | 0xdf            |
| 84d | 22:53:52 | 0x00700000 | 0xef            |
| 91d | 00:32:00 | 0x00780000 | 0xff (reserved) |

Figure 22

Authors' Addresses

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: cabo@tzi.org

Klaus Hartke  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63905  
Email: hartke@tzi.org



CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 17, 2013

C. Bormann  
K. Hartke  
Universitaet Bremen TZI  
July 16, 2012

Congestion Control Principles for CoAP  
draft-bormann-core-congestion-control-01

Abstract

The CoAP protocol needs to be implemented in such a way that it does not cause persistent congestion on the network it uses. Congestion control is a complex issue -- the proper rationale for the congestion control mechanisms chosen in CoAP is probably more material than the CoAP protocol specification itself. This informational document attempts to pull out the background material and more extensive considerations behind the CoAP congestion control mechanisms, while leaving the basic MUSTs and MUST NOTs in the main spec.

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 January 17, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .                            | 3  |
| 1.1. Terminology . . . . .                           | 3  |
| 1.2. Objectives . . . . .                            | 4  |
| 1.2.1. TCP-Friendliness . . . . .                    | 4  |
| 1.2.2. Actually working well . . . . .               | 4  |
| 1.2.3. Getting actual implementation . . . . .       | 5  |
| 2. Input . . . . .                                   | 6  |
| 2.1. RFC 2914 . . . . .                              | 6  |
| 2.2. RFC 5405 . . . . .                              | 6  |
| 2.3. draft-eggert-core-congestion-control . . . . .  | 7  |
| 3. coap-11 Congestion Control Principles . . . . .   | 8  |
| 4. How do other protocols do it . . . . .            | 11 |
| 4.1. DNS . . . . .                                   | 11 |
| 4.2. SIP . . . . .                                   | 11 |
| 4.3. TCP . . . . .                                   | 11 |
| 4.4. HTTP . . . . .                                  | 12 |
| 5. Advanced CoAP Congestion Control . . . . .        | 14 |
| 5.1. RTT Measurement . . . . .                       | 14 |
| 5.2. Block Slow-Start . . . . .                      | 14 |
| 6. Changes Planned for Base Specifications . . . . . | 16 |
| 7. IANA Considerations . . . . .                     | 17 |
| 8. Security Considerations . . . . .                 | 18 |
| 9. Acknowledgements . . . . .                        | 19 |
| 10. References . . . . .                             | 20 |
| 10.1. Normative References . . . . .                 | 20 |
| 10.2. Informative References . . . . .               | 20 |
| Authors' Addresses . . . . .                         | 22 |



## 1. Introduction

With few exceptions, it is simply incompetent to build an implementation of a packet-based protocol without considering congestion control. Unfortunately, detailed, evidence-based knowledge about congestion control is limited to a small group of people. It has become customary for these to try to encode their knowledge into the protocol definitions, in an attempt to replace competence by conformance.

This has worked relatively well for TCP, not the least because the art of TCP implementation is itself limited to a rather small group of experts, which over the years often have acquired some knowledge of congestion control principles, complementing the desire for conformance by substantial competence again. Conversely, application developers are a much larger, much more diverse group. Worse, protocol complexity for which the rationale is not apparent to the developers might simply not be implemented. Giving congestion-unaware developers UDP sockets that are not protected by TCP's congestion control may lead to disasters.

With this background, an application protocol that is threatening to be widely deployed and does not rely on the built-in congestion control properties of TCP presents a serious worry.

This document attempts to present a more extensive rationale for CoAP's minimal, but effective congestion control design, as well as some updates to it. This rationale is not included in [I-D.ietf-core-coap] or [I-D.ietf-core-observe] as the specification is threatening to become too long with all the rationale and implementation considerations discussion already included. While the present document discusses normative statements, it is not intended to supplement or replace the normative statements in [I-D.ietf-core-coap] and [I-D.ietf-core-observe], but just to provide additional explanation.

((Editorial note: the updates to the mandates discussed here partially still need to make it into the next version of [I-D.ietf-core-coap]. A summary of the updates needed is in Section 6.))

### 1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

(Note that this document is itself informational, but it is discussing normative statements.)

The term "byte" is used in its now customary sense as a synonym for "octet".

## 1.2. Objectives

The objectives of adding congestion control to the CoAP protocol specification can be on two different levels, with one additional (third) consideration.

### 1.2.1. TCP-Friendliness

Much of the knowledge that the IETF has accumulated on congestion control focuses on not being worse than its flagship transport protocol, TCP, and being "fair" to instances of TCP competing for capacity. Since fairness is not really a well-defined term, we reduce it to "friendliness".

One objective of this document is to discuss how CoAP can be employed in a TCP-friendly way, and what are the minimum mandates the protocol needs to make in order to ensure this for reasonable applications.

(Note that TCP itself is not TCP-friendly when abused, e.g., when opening 10000 connections in close succession; so there will be no attempt to stay TCP-friendly when CoAP is abused, either.)

Conclusion: CoAP needs to be TCP-friendly, but probably not more so than TCP itself.

### 1.2.2. Actually working well

Making sure that the network continues to work well in the presence of a strong deployment of active CoAP endpoints is a much harder objective to achieve. There is only limited knowledge about the characteristics of the constrained node/networks CoAP will be used in. They might exhibit congestion in surprising ways.

It may turn out the collected wisdom that has been derived from TCP deployment experience in the mostly browser-oriented Internet does not transfer to the Internet of Things, and that we need to invent new mechanisms for the latter.

But this is research.

Imposing the need for a completed solution that meets requirements entirely unknown at this time would be an instance of the Fallacy of

Perfection [GF].

We will need to accumulate additional knowledge, on a research basis, and with experience coming in from larger CoAP deployments. One likely outcome is that constrained node/networks will simply continue to evolve to be able to cope with TCP and CoAP.

Conclusion: For now, we will focus on staying safe where TCP would have stayed safe.

#### 1.2.3. Getting actual implementation

The protocol specification may specify whatever it wants; if there is significant complexity in implementing a mandate and the rationale is not apparent for implementers, compliance will be but a lucky coincidence - even more so in implementations for highly constrained systems. A design that achieves stable operation outside pathological situations and is implemented is preferable to a picture-perfect design that is a beautiful part of the specification and then ignored.

Binding the inevitable complexity of a congestion control scheme to mechanisms that already have to be implemented for other functional reasons seems the most fruitful approach for obtaining compliance. This consideration, together with the main design objective of CoAP - being implementable on constrained nodes and networks - has been the overriding design objective.

## 2. Input

The word "congestion" occurs more than a hundred times in lid-abstracts.txt, indicating that there is a lot of documents under construction that might become relevant to this document. We select a few existing documents here and pick up a few salient points.

### 2.1. RFC 2914

[RFC2914], "Congestion Control Principles", is the BCP that lays out the basic principles for congestion control in the Internet. While it does allude to non-TCP protocols, it mainly focuses on TCP and TCP-like behavior.

### 2.2. RFC 5405

[RFC5405], "Unicast UDP Usage Guidelines for Application Designers", makes additional points for the usage of UDP. It is also a BCP document. Its considerations have mostly been made without looking at specific application protocols, and with a view to guiding application protocol developers towards congestion-controlled transport protocols (which is unfortunately not an appropriate choice for CoAP). It does consider the case of low data-volume applications (section 3.1.2 is therefore the most relevant section for this document). It clearly needs to be interpreted intelligently in order to arrive at congestion control guidelines for a new application protocol. E.g., it recommends:

Applications that at any time exchange only a small number of UDP datagrams with a destination SHOULD still control their transmission behavior by not sending on average more than one UDP datagram per round-trip time (RTT) to a destination.

Instead, a CoAP client that does receive a response without the need for a retransmission should be able to send an ensuing request right away, without the need to do any such rate control -- this keeps the spirit, but not the letter of that requirement.

While [RFC5405] does provide a good set of "don't forget" points, some of its requirements appear to attempt to err on the side of caution, without regards to the specific characteristics of an application. Fortunately, these requirements are often phrased as a SHOULD, so it is possible to explain when and why they should not be heeded.

### 2.3. draft-eggert-core-congestion-control

[I-D.eggert-core-congestion-control], "Congestion Control for the Constrained Application Protocol (CoAP)", was the original document that led to CoAP's congestion control design. This document provides good historical context and should be read in conjunction with the present document. However, the "credit-based" mechanism proposed in its section 3.2 is probably too complicated to be implemented in constrained nodes; CoAP now uses a simpler algorithm that uses the information the implementation already has to keep (i.e., it is based on limiting the outstanding exchanges).

### 3. coap-ll Congestion Control Principles

CoAP is a protocol that attempts to minimize the complexity of its implementation. It is mainly intended for interactions that are not really flow-shaped, so traditional congestion control mechanisms simply do not have useful information to work on.

Basic CoAP [I-D.ietf-core-coap] uses a strict lock-step protocol for its requests and responses (both on the reliability layer with CON/ACK and one level higher with requests and responses), with exponential back-off in case of non-delivery. The initial timeout is dithered between 2 and 3 seconds and grows up to between 32 and 48 seconds.

This is inherently TCP-friendly, similar to the way protocols like DNS operate.

[I-D.ietf-core-coap] goes on to require:

In order not to cause congestion, Clients (including proxies) SHOULD strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies). An outstanding interaction is either a CON for which an ACK has not yet been received but is still expected (message layer) or a request for which a response has not yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). A good value for this limit is the number 1. (Note that [RFC2616], in trying to achieve a similar objective, did specify a specific number of simultaneous connections as a ceiling. While revising [RFC2616], this was found to be impractical for many applications [I-D.ietf-httpbis-pl-messaging]. For the same considerations, this specification does not mandate a particular maximum number of outstanding interactions, but instead encourages clients to be conservative when initiating interactions.)

The rationale for this design is that it is very easy to implement for a constrained device: a constrained device will already have a hard limit on the number of slots available for initiating transactions. Similarly, even back-end systems already need to bind state to outstanding transactions; adding some form of congestion control state to these does not require maintaining new objects, just new fields. In any case, having some form of limit is not elective: in the text the SHOULD needs to be changed into a MUST, even though it may not be easy to pinpoint the exact criterion for compliance.

In the following, we refer to the initiator parameter that limits the number of outstanding interactions as NSTART.

Clients SHOULD also heed this [RFC5405] guideline:

an application SHOULD perform congestion control over all UDP traffic it sends to a destination, independently from how it generates this traffic. For example, an application that forks multiple worker processes or otherwise uses multiple sockets to generate UDP datagrams SHOULD perform congestion control over the aggregate traffic.

Note that [RFC5405] is not explicit here with respect to what it considers to be a "destination"; it also uses the term "destination host" when it appears to provide specific discussion about all protocol entities at an IP address. [RFC5405] duly notes the failure of the congestion manager approach [RFC3124], but appears to wish it back into existence. For the purposes of CoAP, probably "destination" here should be used as with the CoAP term destination endpoint (i.e., including the UDP port number). Still, an implementation that e.g. uses a new source port per request (i.e. a new source endpoint, which is a valid strategy) probably needs to heed this SHOULD for the entirety of the combination of its own endpoint abstractions.

For certain exchanges in CoAP, there is a chance that a request would never elicit a response (e.g., due to a crashed server) but there is also no (protocol) timeout governing this exchange. Therefore, the count of outstanding interactions needs to decay at some rate; a decay rate below that at which TCP sends to a very lossy channel (e.g., 7 B/s) should be safe.

There are also some special congestion control considerations with responses to multicast requests, see [I-D.ietf-core-coap] section 4.5; servers are expected to provide estimates for group size and a target rate as well as a response size. Where those estimates are hard to come up with, a default response dithering window of 10 seconds should be added to [I-D.ietf-core-coap], as well an admonition for a client not to use multicast requests when such a default window would be way off. Finally, a server that receives another multicast request within the dithering window for a request that it already is answering SHOULD move the dithering window for its next response to after the first dithering window.

Finally, the text in [I-D.ietf-core-coap] needs to be reviewed whether it always clearly separates the discussion for avoiding network congestion from any mechanisms for avoiding server overloading.

[I-D.ietf-core-observe] adds one additional behavior: servers may send NON messages as notifications for state changes, which is

outside of exchanges that would be governed by NSTART. This functionality needs to be supported with some discussion of congestion control. Generally, servers SHOULD NOT send more than one NON message every 3 seconds on average ([RFC5405] section 3.1.2), and they SHOULD NOT send NON messages while waiting for CON messages to be acknowledged (however, CON retransmissions should send the new resource state if it has changed since, see [I-D.ietf-core-observe] section 4.5). There already was a decision to add a requirement to require sending a CON message at least every 24 hours before continuing with NON messages; probably the parameter of no more than a NON per 3 seconds should be increased for servers that check the client that rarely (e.g., to the rate at which TCP sends into a very lossy channel, e.g., 7 B/s).



#### 4. How do other protocols do it

While CoAP congestion control could be designed from first principles, it is maybe more realistic to have a look at how other protocols address its respective version of the problem.

##### 4.1. DNS

The DNS protocol, which in many characteristics is quite close to CoAP, does not have any explicit mechanisms for congestion control at all. Many documents consider DNS to be "sporadic messages", not worth of congestion control.

[RFC4336] says:

(The short flows generated by request-response applications, such as DNS and SNMP, don't cause congestion in practice, and any congestion control mechanism would take effect between flows, not within a single end- to-end transfer of information.)

(This simple packet-for-packet request-response nature is now changing a bit with DNS being used for voluminous keying information and growing TXT records.)

##### 4.2. SIP

SIP uses a 0.5 s initial timeout (T1 "RTT Estimate"), and uses binary exponential increase after that. That is similar to CoAP, but starts from a smaller initial estimate. CoAP is more conservative (initial RESPONSE\_TIMEOUT is 2 s to 3 s) as we expect latencies in constrained networks to be higher than in the networks used for telephony.

##### 4.3. TCP

A well-known problem with relying on TCP's built-in congestion control is that, even with all congestion-control mechanisms in place, simply multiplying the number of instances may lead to eventual congestion.

TCP has increased its initial congestion window (IW) to about 3 packets [RFC3390] and is now moving to an IW of 10 packets (IW10) [I-D.ietf-tcpm-initcwnd]. A related change is also planned in that document that will avoid resetting this initial window when the SYN or SYN/ACK is lost. This means that it is considered appropriate to send about 15 kB of data on a single connection without any congestion control feedback whatsoever, except that some SYN+SYN/ACK exchange made it through. While [I-D.ietf-tcpm-initcwnd] is not yet approved, it is a WG document and there is widespread feeling of its

inevitability.

The number 10 clearly provides some additional context for the selection of appropriate values of NSTART.

Conclusion: For now, it is probably appropriate to RECOMMEND keeping NSTART at or below the value 10.

#### 4.4. HTTP

HTTP is running on top of TCP, so it is TCP-friendly by definition. However, as HTTP 1.0 was using one TCP connection per request, and it became clear that browser usage would entail fetching many objects in parallel, congestion was still observed, and client implementations started to limit the number of simultaneously active connections to one server. Even when persistent connections were added (and later codified in HTTP 1.1) this remained a concern. Under 8.1.4 "Practical considerations", [RFC2616] defines a limit on the number of simultaneous connections from one client to one server.

Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy. A proxy SHOULD use up to  $2*N$  connections to another server or proxy, where  $N$  is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion.

Intended as a guideline, this has been implemented to the letter in browser clients for a decade. However, using this as a hard limit is simply not appropriate for all environments. This led server implementers to widely deploy workarounds, such as splitting up a website between multiple servers ("domain sharding") in order to increase the connection concurrency.

From this historical evidence we can learn that well-meaning limitations can cause a lot of pain when implemented slavishly. The httpbis effort has learned this lesson and removed the suggestion for a hard limit (see [HTTPBIS131], [HTTPBIS715]). Note that it now says:

Clients (including proxies) SHOULD limit the number of simultaneous connections that they maintain to a given server (including proxies).

Previous revisions of HTTP gave a specific number of connections as a ceiling, but this was found to be impractical for many applications. As a result, this specification does not mandate a

particular maximum number of connections, but instead encourages clients to be conservative when opening multiple connections.

In particular, while using multiple connections avoids the "head-of- line blocking" problem (whereby a request that takes significant server-side processing and/or has a large payload can block subsequent requests on the same connection), each connection used consumes server resources (sometimes significantly), and furthermore using multiple connections can cause undesirable side effects in congested networks.

Note that servers might reject traffic that they deem abusive, including an excessive number of connections from a client.

Conclusion: There is no doubt that CoAP should follow this hard-learned expertise.

## 5. Advanced CoAP Congestion Control

### 5.1. RTT Measurement

For an initiator that plans to make multiple requests to one destination end-point, it may be worthwhile to make RTT measurements in order to obtain a better RTT estimation than that implied by the default initial timeout of 2 to 3 s. The usual algorithms for RTT estimation can be used [RFC6298], with appropriately extended default/base values. Note that such a mechanism **MUST**, during idle periods, decay RTT estimates that are shorter than the basic RTT estimate back to the basic RTT estimate, until fresh measurements become available again.

One important consideration not relevant for TCP is the fact that a CoAP round-trip may include application processing time, which may be hard to predict, and may differ between different resources available at the same endpoint. Servers will only trigger early ACKs (with a non-piggybacked response to be sent later) based on the default timers, e.g. after 1 s. A client that has arrived at a RTT estimate much shorter than the 2 to 3 s used as a default **SHOULD** therefore not expend all of its retransmissions in the shorter estimated timescale.

It may also be worthwhile to do RTT estimates not just based on information measured from a single destination endpoint, but also based on entire hosts (IP addresses) and/or complete prefixes (e.g., maintain an RTT estimate for a whole /64). The exact way this can be used to reduce the amount of state in an initiator is for further study.

### 5.2. Block Slow-Start

The CoAP protocol is not optimized for making good use of available network capacity; given a good offered load, a lightly-loaded network and some time, a TCP connection will always overtake a series of CoAP requests.

However, the [I-D.ietf-core-block] protocol can be used by inventive clients to emulate TCP slow start. E.g., a client can do a request for block 0, and, if a response comes back without a loss, it can fire off the requests for block 1 and block 2 at the same time, etc., using each response in a similar way that TCP would clock its data segments based on ACKs, waiving NSTART. Similar approaches may work to increase channel utilization for any other REST usage that requires multiple requests.

Clearly, the slow start period **MUST** terminate on the first loss/retransmission. How exactly the congestion window is to be

maintained after that (a "congestion avoidance period" for CoAP) is a subject for further study. See also [I-D.mathis-tcpm-tcp-laminar] for fresh approaches to maintaining the necessary variables in TCP. Another alternative would be an implementation that emulates [RFC5348].

## 6. Changes Planned for Base Specifications

[I-D.ietf-core-coap]:

1. Change SHOULD in second paragraph of [I-D.ietf-core-coap] 4.7 to MUST; define protocol parameter NSTART.
2. Add reference to (and/or cite) [RFC5405] guideline about combining congestion control state for a destination; clarify its meaning for CoAP using the definition of an endpoint.
3. Add a mechanism of decaying outstanding transactions at a rate of about 7 B/s.
4. Add default "Leisure" (response dithering windows) of 10 seconds to [I-D.ietf-core-coap] 8.2, as well as an admonition for a client not to use multicast requests when such a default window would be way off.

[I-D.ietf-core-observe]:

1. servers SHOULD NOT send more than one NON notification every 3 seconds to an endpoint on average ([RFC5405] section 3.1.2); define protocol parameter MAXNONRATE.
2. servers SHOULD NOT send NON messages while waiting for CON messages to be acknowledged (however, CON retransmissions should send the new resource state if it has changed since, see [I-D.ietf-core-observe] section 4.5).
3. require sending a CON message at least every 24 hours before continuing with NON messages.
4. consider increasing MAXNONRATE for servers that check the client that rarely (e.g., to the rate at which TCP sends into a very lossy channel, e.g., 7 B/s).

Additional changes have been made to limit the leeway that implementations have in changing the CoRE protocol parameters; these changes are already gathered in Section 4.8 of [I-D.ietf-core-coap] and will not be repeated here.

## 7. IANA Considerations

This document makes no requirements on IANA. (This section to be removed by RFC editor.)

## 8. Security Considerations

(TBD. The security considerations of, e.g., [RFC2581], [RFC2914], and [RFC5405] apply. Some issues are already discussed in the security considerations of [I-D.ietf-core-coap].)



## 9. Acknowledgements

The first document to examine CoAP congestion control issues in detail was [I-D.eggert-core-congestion-control], to which this draft owes a lot.

Michael Scharf did a review of CoAP congestion control issues that asked a lot of good questions that this draft attempts to answer.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

### 10.2. Informative References

- [GF] Bormann, C., "Garrulity and Fluff", IAB Smart Object Workshop, 2011, <<http://www.iab.org/wp-content/IAB-uploads/2011/04/Bormann.pdf>>.
- [HTTPBISc715] "Changeset 715", October 2009, <<http://trac.tools.ietf.org/wg/httpbis/trac/changeset/715>>.
- [HTTPBISt131] "increase connection limit", HTTPBIS ticket #131, closed 2009-12-02, September 2008, <<http://trac.tools.ietf.org/wg/httpbis/trac/ticket/131>>.
- [I-D.eggert-core-congestion-control] Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [I-D.ietf-core-block] Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-08 (work in progress), February 2012.
- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-11 (work in progress), July 2012.
- [I-D.ietf-core-observe] Hartke, K., "Observing Resources in CoAP",

draft-ietf-core-observe-05 (work in progress), March 2012.

[I-D.ietf-httpbis-pl-messaging]

Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part 1: Message Routing and Syntax", draft-ietf-httpbis-pl-messaging-20 (work in progress), July 2012.

[I-D.ietf-tcpm-initcwnd]

Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", draft-ietf-tcpm-initcwnd-04 (work in progress), June 2012.

[I-D.mathis-tcpm-tcp-laminar]

Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", draft-mathis-tcpm-tcp-laminar-00 (work in progress), February 2012.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[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, June 1999.

[RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

[RFC4336] Floyd, S., Handley, M., and E. Kohler, "Problem Statement for the Datagram Congestion Control Protocol (DCCP)", RFC 4336, March 2006.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

Authors' Addresses

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: cabo@tzi.org

Klaus Hartke  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63905  
Email: hartke@tzi.org



CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 29, 2013

C. Bormann  
Universitaet Bremen TZI  
February 25, 2013

Representing CoRE Link Collections in JSON  
draft-bormann-core-links-json-02

Abstract

Web Linking (RFC5988) provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format (RFC6690). Outside of constrained environments, it may be useful to represent these collections of Web links in JSON format (RFC4627).

This specification defines a common format for representing Web links in JSON format.

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 August 29, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|                                       |   |
|---------------------------------------|---|
| 1. Introduction . . . . .             | 2 |
| 1.1. Objectives . . . . .             | 2 |
| 1.2. Terminology . . . . .            | 3 |
| 2. Web Links in JSON . . . . .        | 3 |
| 2.1. Examples . . . . .               | 4 |
| 3. IANA Considerations . . . . .      | 5 |
| 4. Security Considerations . . . . .  | 5 |
| 5. Acknowledgements . . . . .         | 5 |
| 6. References . . . . .               | 5 |
| 6.1. Normative References . . . . .   | 5 |
| 6.2. Informative References . . . . . | 5 |
| Appendix A. Implementation . . . . .  | 6 |
| Author's Address . . . . .            | 6 |

## 1. Introduction

Web Linking [RFC5988] provides a way to represent links between Web resources as well as the relations expressed by them and attributes of such a link. In constrained networks, a collection of Web links can be exchanged in the CoRE link format [RFC6690] to enable resource discovery.

Outside of constrained environments, it may also be useful to represent the same collections of Web links in the widely used JSON format [RFC4627]. When converting between these two formats, as usual, there are many little decisions that have to be made. If left without guidance, it is likely that a number of slightly incompatible dialects will emerge.

This specification defines a common format for representing CoRE Web Linking in JSON format.

Note that there is a separate question on how to represent Web links out of JSON documents, as discussed e.g. in [MNOT11]. While there are good reasons to stay as compatible as possible to developments in this area, the present specification is solving a different problem.

### 1.1. Objectives

(TBD: Convert the shopping list into plaintext)

- o Canonical mapping
  - \* lossless round-tripping
  - \* but not trying for bit-preserving (DER-style) round-tripping
- o The simplest thing that could possibly work
  - \* Do not cater for RFC 5988 complications caused by HTTP header character set issues [RFC2047]
- o Consider other work that has links in JSON, e.g.: JSON-LD, JSON-Reference [I-D.pbryan-zyp-json-ref]
  - \* Do not introduce unmotivated differences

## 1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

## 2. Web Links in JSON

The objective of the JSON mapping defined in this document is to contain information of the formats specified in [RFC5988] and [RFC6690]. This specification therefore uses the names of the ABNF productions used in those documents.

An application/link-format document is a collection of web links ("link-value"), each of which is a collection of attributes ("link-param") applied to a "URI-Reference".

We straightforwardly map:

- o the outer collection to an array of links
- o each link to a JSON object.

In the object representing a "link-value", each target attribute or other parameter ("link-param") is represented by a JSON name/value pair (member). The name is a string representation of the parameter or attribute name (as in "paramname"), the value is a string representation of the parameter or attribute value ("ptoken" or "quoted-string"). "quoted-string" productions are parsed (i.e, the backslash constructions evaluated) as defined in [RFC6690] and its



referenced documents, before placing them in JSON strings (where they may gain back additional decorations such as backslashes as defined in [RFC4627]).

If a Link attribute ("parmname") is present more than once in a "link-value", its values are then represented as a JSON array of JSON string values; this array becomes the value of the JSON name/value pair where the attribute name is the JSON name. Attributes occurring just once MUST NOT be represented as JSON arrays but MUST be directly represented as JSON strings. (Note that the most recent version of link-format has cut down on the use of repeated parameter names; they are still allowed by [RFC5988] though. No attempt has been made to decode the possibly space-separated values for rt=, if=, and rel= into JSON arrays.)

The URI-Reference is represented as a name/value pair with the name "href" and the URI-Reference as the value. (Rationale: This usage is consistent with the use of "href" as a query parameter for link-format query filtering and with link-format reserving the link parameter "href" specifically for this use [RFC6690]).

(TBD: Should we do something special with the "hosts" relation? Should we include an anchor where the link-format does not explicitly set one?)

## 2.1. Examples

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
<http://www.example.com/sensors/tl23>;anchor="/sensors/temp"
;rel="describedby",
</t>;anchor="/sensors/temp";rel="alternate"
```

Figure 1: Example from page 15 of [RFC6690]

becomes

```
"[{\"href\":\"/sensors\",\"ct\":\"40\",\"title\":\"Sensor Index\"},{\"href\":\"/sensors/temp\",\"rt\":\"temperature-c\",\"if\":\"sensor\"},{\"href\":\"/sensors/light\",\"rt\":\"light-lux\",\"if\":\"sensor\"},{\"href\":\"http://www.example.com/sensors/tl23\",\"anchor\":\"/sensors/temp\",\"rel\":\"describedby\"},{\"href\":\"/t\",\"anchor\":\"/sensors/temp\",\"rel\":\"alternate\"}] "
```

(More examples to be added.)

### 3. IANA Considerations

(TBD. All the Media Type boilerplate, too, for:)

application/link-format+json

### 4. Security Considerations

(TBD.)

### 5. Acknowledgements

(TBD.)

### 6. References

#### 6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

#### 6.2. Informative References

- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.pbryan-zyp-json-ref] Bryan, P. and K. Zyp, "JSON Reference", draft-pbryan-zyp-json-ref-03 (work in progress), September 2012.
- [MNOT11] Nottingham, M., "Linking in JSON", November 2011, <[http://www.mnot.net/blog/2011/11/25/linking\\_in\\_json](http://www.mnot.net/blog/2011/11/25/linking_in_json)>.
- [RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

## Appendix A. Implementation

This appendix provides a simple reference implementation of the mapping between CoRE link format and Links-in-JSON.

(TBD - the reference implementation was used to create the above examples, but I still have to clean it up for readability and paste it in at 69 columns max.)

## Author's Address

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: cabo@tzi.org

CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: April 25, 2014

C. Bormann  
Universitaet Bremen TZI  
October 22, 2013

CoRE Roadmap and Implementation Guide  
draft-bormann-core-roadmap-05

Abstract

The CoRE set of protocols, in particular the CoAP protocol, is defined in draft-ietf-core-coap in conjunction with a number of specifications that are currently nearing completion. There are also several dozen more individual Internet-Drafts in various states of development, with various levels of WG review and interest.

Today, this is simply a bewildering array of documents. Beyond the main four documents, it is hard to find relevant information and assess the status of proposals. At the level of Internet-Drafts, the IETF has only adoption as a WG document to assign status - too crude an instrument to assess the level of development and standing for anyone who does not follow the daily proceedings of the WG.

With a more long-term perspective, as additional drafts mature and existing specifications enter various levels of spec maintenance, the entirety of these specifications may become harder to understand, pose specific implementation problems, or be simply inconsistent.

The present guide aims to provide a roadmap to these documents as well as provide specific advice how to use these specifications in combination. In certain cases, it may provide clarifications or even corrections to the specifications referenced.

This guide is intended as a continued work-in-progress, i.e. a long-lived Internet-Draft, to be updated whenever new information becomes available and new consensus on how to handle issues is formed. Similar to the ROHC implementation guide, RFC 4815, it might be published as an RFC at some future time later in the acceptance curve of the specifications.

This document does not describe a new protocol or attempt to set a new standard of any kind - it mostly describes good practice in using the existing specifications, but it may also document emerging consensus where a correction needs to be made.

(TODO: The present version does not completely cover the new Internet-Drafts submitted concurrently with it; it is to be updated by the start of IETF88.)

#### Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2014.

#### Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

#### Table of Contents

|  |   |
|--|---|
| 1. Introduction . . . . .                        | 3 |
| 1.1. Terminology . . . . .                       | 3 |
| 2. The Main Four . . . . .                       | 3 |
| 2.1. The CoAP protocol . . . . .                 | 4 |
| 2.2. Discovery . . . . .                         | 5 |
| 2.3. Further reading . . . . .                   | 6 |
| 3. Informational Drafts . . . . .                | 6 |
| 3.1. Implementation . . . . .                    | 6 |
| 3.2. Multicast and Group Communication . . . . . | 7 |
| 3.3. Security . . . . .                          | 8 |

|       |  |    |
|-------|--|----|
| 3.4.  | Intermediaries . . . . .                                   | 9  |
| 3.5.  | Congestion Control . . . . .                               | 9  |
| 4.    | CoAP over X . . . . .                                      | 9  |
| 5.    | Optional components of CoRE . . . . .                      | 10 |
| 5.1.  | CoAP-misc . . . . .  | 10 |
| 5.2.  | Generalizing Media Types . . . . .                         | 11 |
| 5.3.  | Patience, Leisure, Pledge, or: Timing extensions . . . . . | 11 |
| 5.4.  | Extending Observe . . . . .                                | 11 |
| 5.5.  | Service discovery . . . . .                                | 11 |
| 5.6.  | Server discovery, Naming, etc. . . . .                     | 12 |
| 5.7.  | More support for sleepy nodes . . . . .                    | 12 |
| 6.    | Replaced drafts . . . . .                                  | 14 |
| 7.    | IANA Considerations . . . . .                              | 15 |
| 8.    | Security Considerations . . . . .                          | 15 |
| 9.    | Acknowledgements . . . . .                                 | 15 |
| 10.   | References . . . . .                                       | 15 |
| 10.1. | Normative References . . . . .                             | 15 |
| 10.2. | Informative References . . . . .                           | 16 |
|       | Author's Address . . . . .                                 | 22 |

## 1. Introduction

(To be written - for now please see the Abstract.)

### 1.1. Terminology

This document is a guide. However, it might evolve to make specific recommendations on how to use standards-track specifications. Therefore: 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. They indicate requirement levels for compliant CoRE implementations [RFC2119]. Note that these keywords are not only used where a correction or clarification is intended; the latter are explicitly identified as such.

The term "byte" is used in its now customary sense as a synonym for "octet".

## 2. The Main Four

The main component of the CoRE architecture is the Constrained Application Protocol (CoAP). It aims to provide a RESTful transfer service, not unlike HTTP, but radically simplified for the use on constrained devices on constrained networks. REST is the architectural style that informed the design of HTTP [REST]. The terms "constrained device" and "constrained network" refer to limited-capability devices such as sensors operating on networks such

as the IEEE 802.15.4 based 6LoWPAN [RFC4919].

[I-D.ietf-lwig-terminology] provides a more detailed discussion of what we mean by these terms.

## 2.1. The CoAP protocol

The CoAP protocol is defined in three specifications:

- o [I-D.ietf-core-coap]
- o [I-D.ietf-core-block]
- o [I-D.ietf-core-observe]

The first specification, [I-D.ietf-core-coap], provides the core transfer protocol, including the means to provide communication security using the DTLS protocol [RFC6347] (compare this to the way [RFC2616] and [RFC2818] define HTTP and HTTPS). The protocol is structured into a message layer, which provides duplicate detection and optional message reliability on top of UDP, and a request/response layer, which provides the usual REST operations GET, PUT, POST, and DELETE. A highly efficient protocol encoding carries the 4-byte base header, a sequence of `_Options_`, and the payload (body) of a message. The main extension points of CoAP are its Options, similar to the way new header fields are used to extend HTTP.

Since CoAP is a very simple protocol running on top of UDP, it is limited in its transfer size by the datagram sizes provided by UDP. As a further constraint, many constrained networks do not provide good reliability of delivery once their small frame sizes are exceeded and the adaptation layer is forced to fragment [WEI]. This may lead to a practical limitation to payload sizes as small as 64 bytes. [I-D.ietf-core-block] extends the base CoAP protocol with three options that enable `_blockwise_` transfer, i.e., splitting up a larger transfer into a sequence of smaller transactions, as well as the early determination of the overall size of the resource representation.

In HTTP, transactions are always client initiated, and it is the responsibility of the client to perform GET operations again and again (polling) if it wants to stay up to date about the status of a resource. This "pull model" becomes expensive in an environment with limited power, limited network resources, and nodes that sleep most of the time. Some more or less savory workarounds have been developed for HTTP [RFC6202], but, as a new protocol, CoAP can do better. [I-D.ietf-core-observe] extends the base CoAP protocol with an option that a client can use to indicate its interest in further updates from a resource. If the server accepts this option, the

client becomes an `_Observer_` of this resource and receives an asynchronous notification message each time it changes. Each such notification message is identical in structure to the response to the initial GET request.

While the "Block" and "Observe" specifications are optional additions to the CoAP protocol (just as the core specification already defines 14 options most of which will not need to be used in every message), they together form what is now generally considered to be the CoAP protocol.

The CoRE Working Group has completed its work on the base CoAP protocol specification [I-D.ietf-core-coap] and it has been approved by the IESG for publication as a Standards-Track RFC on 2013-07-15. The completed document is currently waiting in the RFC editor queue for two of its normative references in the security area, [I-D.mcgregw-tls-aes-ccm-ecc] and [I-D.ietf-tls-oob-pubkey], to be completed and approved.

The other two CoAP specifications are, at the time of this writing, in the process of being updated based on the comments to the first Working-Group Last-Call [RFC2418], and in the second Working-Group Last-Call, respectively; these are prerequisites to submitting them to the IESG for publication as a Standards-Track RFC.

The specifications, together with link-format (below), have been widely implemented in highly interoperable implementations: an ETSI "plugtest" event in March 2012 was attended by 15 organizations with 20 implementations; in over 3000 tests performed only about 6 % failed; a second plugtest was conducted in November 2012 and led to some final adjustments of some details in the specifications. Another plugtest is planned for November 2013 [COAP3].

## 2.2. Discovery

The fourth specification in the main set now nearing completion does not extend the CoAP protocol but addresses a different problem.

In the Web, a number of methods for discovery of resources are common. Initially, Web discovery was just performed by humans based on an entry resource to a server (e.g., `/index.html`). This resource then includes links that directly or indirectly allow a human to reach the other Web resources that make up the Web site.

Web discovery can be performed by machines if standardized interfaces and resource descriptions are available. Among the component mechanisms for Web discovery that are standardized in the IETF are the well-known resource path `"/.well-known/..."` [RFC5785] and the



HTTP link header [RFC5988]. Several related techniques are in common use today.

Clearly, in the machine-to-machine environments that will be typical of CoAP applications, it is important to enable devices to discover each other and their resources. Autonomous devices and embedded systems necessitate uniform, interoperable resource discovery.

A basic component for this is provided by a standardized description format for the resources a server provides, the `_link-format_`. Unless other methods of discovery are available, CoAP servers should provide such a description via the well-known URI `"/.well-known/core"`, available for access via a GET request on that URI. (More advanced resource discovery schemes might make the same description available by other means, e.g. by posting it to a resource directory.)

The description format has been adapted from the format used in the HTTP link header [RFC5988], which is simple and easy to parse. In contrast to the HTTP specification, link-format is specified as an Internet media type (what used to be called "MIME type") and intended to be carried around in the payload [RFC6690].

[RFC6690] was the first RFC of the CoRE working group.

### 2.3. Further reading

A recent article provides a more detailed overview over the CoRE documents nearing completion [SB].

While the specification documents themselves have to go into meticulous details on every aspect of their protocols, they are the ultimate reference source and are the recommended reading if this basic overview is not sufficient.

## 3. Informational Drafts

### 3.1. Implementation

In the IETF, a separate working group is working on informational documents concerning guidance in lightweight implementation of protocols, the LWIG working group. LWIG has several drafts pertinent here:

[I-D.ietf-lwig-terminology] provides some common terms that are useful for discussing implementations and specification in the constrained node network space. Section 2 and 3 of this document are quite stable at this time; a new section 4 is in preparation that

will include discussion of power-related terminology.

[I-D.ietf-lwig-cellular] provides a well-founded discussion of methods for power conservation in CoAP nodes connected via cellular networks, from which some of the material will be used.

[I-D.ietf-lwig-guidance] was originally intended as the main working document of the WG. It contains some discussion about CoAP implementation in its section 3.4.2, including the efficient representation of managing duplicate detection state.

[I-D.kovatsch-lwig-classl-coap] contains additional considerations that, over time, might move into [I-D.ietf-lwig-guidance].

[I-D.castellani-lwig-coap-separate-responses] contains some examples for message exchanges, focusing on elaborating exchanges involving separate responses. Since IETF86, work is under way to merge the CoAP-related information from these three drafts into a new document, [I-D.kovatsch-lwig-coap].

A new working group has been established in the IETF Security Area to address the use of DTLS In Constrained Environments (DICE); several drafts are available for discussion at IETF88 in Vancouver. On the implementation side, two drafts show how to build minimal implementations of security protocols relevant for CoAP:

[I-D.ietf-lwig-tls-minimal] for TLS, which is relevant for CoAP's use of DTLS; and [I-D.ietf-lwig-ikev2-minimal] for IKEv2, the protocol for setting up IPsec security associations. Similarly, [I-D.hartke-core-codtls] looks specifically into the use of DTLS in constrained networks. It raises issues that pertain both to the LWIG and CoRE working groups of the IETF.

Further drafts submitted to LWIG address energy efficient implementation [I-D.hex-lwig-energy-efficient] and recent developments in operating systems for constrained devices [I-D.hahm-lwig-painless-constrained-programming].

After a somewhat slow start, LWIG is now picking up considerable energy.

### 3.2. Multicast and Group Communication

As it is based on UDP, CoAP easily supports the use of IP multicast to confer messages. However, there are difficult issues around making the desirable multicast applications actually work well.

This led to an additional milestone on the CoRE charter:

Nov 2012: Using CoAP for group communications to IESG as Informational

The informational WG draft [I-D.ietf-core-groupcomm] discusses fundamentals and use cases for group communication with CoAP. This is now very close to Working Group last call.

[I-D.dijk-core-groupcomm-misc] gives some additional considerations, listing requirements, providing some taxonomy, proposing deployment guidelines, and discussing approaches that are not (yet?) in the focus of the WG. Its section 5 can serve as an overview over the status of multicast in constrained node/networks.

### 3.3. Security

Several individual drafts analyze the issues around the security of constrained devices in constrained networks.

[I-D.garcia-core-security] in particular describes the "Thing Lifecycle" and discusses resulting architectural considerations.

[I-D.sarikaya-core-secure-bootsolution] documents the approach taken in the ZigBee IP specification (used in Smart Energy Profile 2.0); the CoRE WG currently is not working on replicating this specification as an IETF document.

[I-D.jennings-core-transitive-trust-enrollment] demonstrates a specific approach to securing the Thing Lifecycle based on defined roles of security players, including a Manufacturer, an Introducer, and a Transfer Agent. There is considerable interest in the CoRE working group to complete one or more specifications in this space.

Further work around Thing Lifecycles was expected to occur in the SOLACE initiative (Smart Object Lifecycle Architecture for Constrained Environments), with its early mailing list at [solace@ietf.org](mailto:solace@ietf.org) -- developed after the model of the COMAN initiative (Management for Constrained Management Networks and Devices, [coman@ietf.org](mailto:coman@ietf.org), [I-D.ersue-constrained-mgmt]).

Besides [I-D.garcia-core-security], recently, more work has been focused on the Authentication and Authorization aspects of CoRE:

- o [I-D.gerdes-core-dcaf-authorize]
- o [I-D.greevenbosch-core-authreq]
- o [I-D.pporamba-dtls-certkey]
- o [I-D.urien-core-racs]
- o [I-D.schmitt-two-way-authentication-for-iot]

- o [I-D.seitz-core-sec-usecases]
- o [I-D.selander-core-access-control]
- o [I-D.zhu-core-groupauth]

### 3.4. Intermediaries

[I-D.castellani-core-http-mapping] discusses some ideas about what HTTP/CoAP intermediaries could do beyond the basic mapping defined in [I-D.ietf-core-coap]; in the IETF86 WG meeting, this document was agreed as a future working group item (with validation of the adoption on the mailing list still pending). An earlier version of this draft was split into the current document describing best practices for mapping between HTTP and CoAP (beyond what is already described in [I-D.ietf-core-coap]), and one additional document that describes usages that serve as additional useful examples for more advanced forms of mapping, a first draft of the latter is available in [I-D.castellani-core-advanced-http-mapping].

### 3.5. Congestion Control

[I-D.ietf-core-coap] only defines a very basic congestion control scheme that is focused on being safe in a wide variety of applications. Additional documents will define more advanced congestion control schemes that can provide more optimized performance in exchange for more implementation complexity and/or a narrower field of application.

Several drafts are contributing to this active subject of discussion in the WG:

|                                       |     |            |
|---------------------------------------|-----|------------|
| draft-bormann-core-congestion-control | -02 | 2012-08-01 |
| draft-bormann-core-cocoa              | -00 | 2012-08-13 |

[I-D.greevenbosch-core-minimum-request-interval] proposes adding an option that allows a server to indicate its desire for some pacing of the requests sent to it by one client; enabling a form of server load control.

## 4. CoAP over X

[I-D.becker-core-coap-sms-gprs] shows how to run CoAP over cellular SMS and in mixed SMS/GPRS environments. This draft optionally makes use of an SMS-oriented encoding for CoAP that is described in [I-D.bormann-coap-misc]. [I-D.silverajan-core-coap-alternative-transports] discusses how to indicate the alternative transport in a URI.

[I-D.li-core-coap-payload-length-option] defines a way to indicate the length of the payload in case the underlying transport does not provide a suitable definite length indication.

## 5. Optional components of CoRE

Additional sub-protocols are being discussed in the IETF that may become optional protocols in CoREs.

The present document will track these sub-protocols and be amended once the sub-protocols reach formal status in the IETF.

Since the WG is cautious in adopting additional work while the main specifications near completion, none of the additional protocols proposed have become WG documents yet.

### 5.1. CoAP-misc

One draft is a little different from the other drafts in this category: [I-D.bormann-coap-misc] is a running document capturing CoAP extensions that are in various states of being cooked.

Some of these extensions may finally be adopted for the WG documents and then vanish from CoAP-misc. For other extensions, we may decide that they are not very good ideas. Instead of deleting them from CoAP-misc, they are moved to an appendix. This documents the approach, the best implementation of that approach that was reached, and the reasons why it was not adopted. This documentation should spare the WG and its contributors from the continuous reinvention of bad ideas.

As of the time of writing, the main body of CoAP-misc is almost empty, as most urgent developments have found their way into the WG documents, and many other ideas wait in the "nursery" section of the document.

## 5.2. Generalizing Media Types

CoAP defines a registry for combinations of an Internet Media Type ("MIME type") and a Content Encoding (e.g. some form of compression), enabling its compact encoding of this information in one or two bytes. Each entry in the registry defines a single, fixed set of media type parameters (as in ";charset=utf-8"), if any. This does not work well with media types that rely on more complex combinations of parameter settings. [I-D.doi-core-parameter-option] proposes to add an option to carry parameters for media types.

[I-D.fossati-core-multipart-ct] defines a new media type that can carry multiple embedded representations employing different media types using a binary type-length-value format.

## 5.3. Patience, Leisure, Pledge, or: Timing extensions

Several proposals intend to extend the amount of information available during an exchange about the timing requirements of the participants.

| draft-li-core-coap-patience-option | -01 | 2012-10-22 |

Another discussion is in Appendix B.4 of [I-D.bormann-coap-misc].

The question of whether some of this functionality should be introduced into the main WG documents now is currently also the subject of an active issue tracker ticket [CoRE204].

## 5.4. Extending Observe

## 5.5. Service discovery

Basic service discovery is defined in [RFC6690]. A JSON representation of the same information is defined in [I-D.ietf-core-links-json]. The intention is to make this information available in an equivalent format that is more accessible to classic Web servers, both as a file format (Internet media type) and as a format that can be used in e.g. a JavaScript API.

[I-D.arkko-core-dev-urn] defines a new Uniform Resource Name (URN) namespace that can be used to provide hardware device identifiers in resource descriptions.

[I-D.ietf-core-interfaces] provides additional semantics that can be used to make resource descriptions more directly machine-interpretable. This ties in to a more general discussion about CoRE profiles that has only just begun.

[I-D.greevenbosch-core-profile-description] ties into this and defines a basic JSON format for indicating what CoAP Options and what Content-Formats (still called media-types there) are available for a resource. At IETF86 there was fairly good consensus in the CoRE WG that we should be working on something addressing the underlying problem statement, while there was not yet agreement on the specific solution.

[I-D.fossati-core-fp-link-format-attribute] defines a link-format attribute that indicates a certain resource is best reached via a specific proxy.

#### 5.6. Server discovery, Naming, etc.

On the boundary between service and server discovery, resource directory servers provide a way to collect resource descriptions from multiple servers into one accessible location.

[I-D.bormann-core-simple-server-discovery] provided a basic way to discover such servers in a constrained node/network without necessarily having to resort to multicast. It has been merged into [I-D.ietf-core-resource-directory], which defines protocol elements that can be used for setting up such a resource directory.

An attempt to merge mDNS/DNS-SD-based discovery (colloquially known as zeroconf or Bonjour), including recent approaches to extend these for constrained networks, into the picture is documented in [I-D.vanderstok-core-dna]; at IETF86 the authors showed interest to continue work on this.

#### 5.7. More support for sleepy nodes

The basic communication model of CoAP was imported from the Web. This applies well to some communication requirements in constrained node/networks, but leaves some other requirements open.

The assumption underlying the current set of WG documents is that the communication layers below the application provide support functions for sleeping nodes. Adding support at the application layer might be able to further reduce the power requirements of "sleepy nodes" that can sleep most of the time.

[I-D.rahman-core-sleepy-problem-statement] summarizes the overall problem statement for sleepy nodes without getting into any specific solution.

A number of drafts aim to extend the CoAP communication model towards more support for sleepy nodes.

The base CoAP spec [I-D.ietf-core-coap] already provides some rudimentary support of sleepy nodes by supporting caching in intermediaries: resources from a sleepy node may be available from a caching proxy (if previously retrieved) even though the node is asleep. [I-D.ietf-core-observe] enhances this support by enabling sleepy nodes to update caching intermediaries on their own schedule.

A number of drafts more extensively extend the concept of an intermediary by introducing an additional kind of server that is hosting the resources of the sleepy node:

The approach of [I-D.vial-core-mirror-server] is to store the actual resource representations in a special type of Resource Directory called the Mirror Server. Communicating devices can then fetch the resource from the Mirror Server regardless of the state of the sleepy server. ([I-D.vial-core-mirror-proxy] simply appears to be a previous version of this draft.)

Similar to the above, the approach of [I-D.fossati-core-publish-option] is to temporarily delegate authority of its resources (when it is sleeping) to a proxy server that is always on.

Also, the approach of [I-D.giacomin-core-sleepy-option] is to define a proxy that acts as a store-and-forward agent for a sleepy node.

Other drafts introduce a variety of signaling based approaches to facilitate communicating with sleepy nodes: The approach of [I-D.castellani-core-alive] is to define a new CoAP message type (called "Alive") which the sleepy node multicasts to all interested devices when it wakes up. The approach of [I-D.rahman-core-sleepy] is to introduce storing of sleep characteristics in the Resource Directory. Communicating devices can then query the RD to learn the sleep status of the sleepy node before attempting communications.

Finally, some drafts build on the concept of the Observe mechanism to help keep track of the sleepy node information. The approach of [I-D.fossati-core-monitor-option] is to extend the Observe pattern to handle the scenario when both server and clients are sleepy nodes. Note that some of the other drafts (e.g., [I-D.vial-core-mirror-server], [I-D.rahman-core-sleepy]) include



using/extending the Observe mechanism as part of their overall approach.

Support for sleepy nodes is currently a very active subject of discussion in the WG; it is clear that there is a high level of interest in the WG in addressing application-level support for sleepy nodes in future specifications. See also the discussion of [I-D.ietf-lwig-cellular] in Section 3.1 above.

## 6. Replaced drafts

Internet-Drafts often get replaced by merged drafts or get promoted to WG drafts. As the relationships between drafts are not always accurately captured by the secretariat tools, this table provides a mapping from current drafts to any previous drafts they are replacing:

| current draft                                    | replaced draft                             |
|--|--|
| [I-D.ietf-core-coap]                             | draft-shelby-core-coap                     |
| [I-D.ietf-core-block]                            | draft-bormann-core-coap-block              |
|  | draft-li-core-coap-size-option             |
| [I-D.ietf-core-observe]                          | draft-hartke-coap-observe                  |
| [RFC6690]  | draft-shelby-core-link-format              |
| [I-D.ietf-core-groupcomm]                        | draft-rahman-core-groupcomm                |
| [I-D.becker-core-coap-sms-gprs]                  | draft-li-core-coap-over-sms                |
| [I-D.vanderstok-core-dna]                        | draft-vanderstok-core-bc                   |
| [I-D.ietf-core-resource-directory]               | draft-bormann-core-simple-server-discovery |
| [I-D.greevenbosch-core-minimum-request-interval] | draft-greevenbosch-core-block-minimum-time |

Note that draft-scim-core-schema is just named against the naming conventions and actually unrelated to the CoRE working group.

## 7. IANA Considerations

This document has no actions for IANA.

## 8. Security Considerations

(None so far; this section will certainly grow as additional security considerations beyond those listed in the base specifications become known.)

## 9. Acknowledgements

(The concept for this document is borrowed from [RFC4815], which was invented by Lars-Erik Jonsson. Thanks!)

Akbar Rahman contributed text to this roadmap.

## 10. References

### 10.1. Normative References

- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-13 (work in progress), October 2013.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-11 (work in progress), October 2013.
- [I-D.ietf-tls-oob-pubkey]  
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", draft-ietf-tls-oob-pubkey-10 (work in progress), October 2013.
- [I-D.mcgreew-tls-aes-ccm-ecc]  
McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM ECC Cipher Suites for TLS", draft-mcgreew-tls-aes-ccm-ecc-07 (work in progress), August 2013.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

## 10.2. Informative References

- [COAP3] ETSI plugtests, "CoAP 3 & OMA Lightweight M2M", 2013, <<http://www.etsi.org/coap-oma-lightweight-m2m>>.
- [CoRE204] Bormann, C., "Introduce a minimal version of Pledge", CoRE ticket #204, 2012, <<http://trac.tools.ietf.org/wg/core/trac/ticket/204>>.
- [I-D.arkko-core-dev-urn]  
Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-arkko-core-dev-urn-03 (work in progress), July 2012.
- [I-D.becker-core-coap-sms-gprs]  
Becker, M., Li, K., Poetsch, T., and K. Kuladinithi, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-04 (work in progress), August 2013.
- [I-D.bormann-coap-misc]  
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-25 (work in progress), May 2013.
- [I-D.bormann-core-simple-server-discovery]  
Bormann, C., "CoRE Simple Server Discovery", draft-bormann-core-simple-server-discovery-01 (work in progress), March 2012.
- [I-D.castellani-core-advanced-http-mapping]  
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-advanced-http-mapping-02 (work in progress), July 2013.

- [I-D.castellani-core-alive]  
Castellani, A. and S. Loreto, "CoAP Alive Message", draft-castellani-core-alive-00 (work in progress), March 2012.
- [I-D.castellani-core-http-mapping]  
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-http-mapping-07 (work in progress), February 2013.
- [I-D.castellani-lwig-coap-separate-responses]  
Castellani, A., "Learning CoAP separate responses by examples", draft-castellani-lwig-coap-separate-responses-00 (work in progress), March 2012.
- [I-D.dijk-core-groupcomm-misc]  
Dijk, E. and A. Rahman, "Miscellaneous CoAP Group Communication Topics", draft-dijk-core-groupcomm-misc-04 (work in progress), June 2013.
- [I-D.doi-core-parameter-option]  
Doi, Y. and K. Lynn, "CoAP Content-Type Parameter Option", draft-doi-core-parameter-option-03 (work in progress), August 2013.
- [I-D.ersue-constrained-mgmt]  
Ersue, M., Romascanu, D., and J. Schoenwaelder, "Management of Networks with Constrained Devices: Problem Statement, Use Cases and Requirements", draft-ersue-constrained-mgmt-03 (work in progress), February 2013.
- [I-D.fossati-core-fp-link-format-attribute]  
Fossati, T. and S. Loreto, "Resource Discovery through Proxies", draft-fossati-core-fp-link-format-attribute-00 (work in progress), July 2012.
- [I-D.fossati-core-monitor-option]  
Fossati, T., Giacomini, P., and S. Loreto, "Monitor Option for CoAP", draft-fossati-core-monitor-option-00 (work in progress), July 2012.
- [I-D.fossati-core-multipart-ct]  
Fossati, T., "Multipart Content-Format Encoding for CoAP", draft-fossati-core-multipart-ct-03 (work in progress), October 2013.
- [I-D.fossati-core-publish-option]

Fossati, T., Giacomini, P., and S. Loreto, "Publish Option for CoAP", draft-fossati-core-publish-option-02 (work in progress), October 2013.

[I-D.garcia-core-security]

Garcia-Morchon, O., Kumar, S., Keoh, S., Hummen, R., and R. Struik, "Security Considerations in the IP-based Internet of Things", draft-garcia-core-security-06 (work in progress), September 2013.

[I-D.gerdes-core-dcaf-authorize]

Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authorization Function (DCAF)", draft-gerdes-core-dcaf-authorize-00 (work in progress), July 2013.

[I-D.giacomini-core-sleepy-option]

Fossati, T., Giacomini, P., Loreto, S., and M. Rossini, "Sleepy Option for CoAP", draft-giacomini-core-sleepy-option-00 (work in progress), February 2012.

[I-D.greevenbosch-core-authreq]

Greevenbosch, B., "Use cases and requirements for authentication and authorisation in CoAP", draft-greevenbosch-core-authreq-00 (work in progress), September 2013.

[I-D.greevenbosch-core-minimum-request-interval]

Greevenbosch, B., "CoAP Minimum Request Interval", draft-greevenbosch-core-minimum-request-interval-01 (work in progress), April 2013.

[I-D.greevenbosch-core-profile-description]

Greevenbosch, B., Hoebeke, J., Ishaq, I., and F. Abeele, "CoAP Profile Description Format", draft-greevenbosch-core-profile-description-02 (work in progress), June 2013.

[I-D.hahm-lwig-painless-constrained-programming]

Hahm, O., Baccelli, E., and K. Schleiser, "Painless Class 1 Devices Programming", draft-hahm-lwig-painless-constrained-programming-00 (work in progress), March 2013.

[I-D.hartke-core-codtls]

Hartke, K. and O. Bergmann, "Datagram Transport Layer Security in Constrained Environments", draft-hartke-core-codtls-02 (work in progress), July 2012.

[I-D.hex-lwig-energy-efficient]

Cao, Z., He, X., Kovatsch, M., Tian, H., and C. Gomez,  
"Energy Efficient Implementation of IETF Constrained  
Protocol Suite", draft-hex-lwig-energy-efficient-02 (work  
in progress), October 2013.

[I-D.ietf-core-groupcomm]

Rahman, A. and E. Dijk, "Group Communication for CoAP",  
draft-ietf-core-groupcomm-16 (work in progress), October  
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-links-json]

Bormann, C., "Representing CoRE Link Collections in JSON",  
draft-ietf-core-links-json-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.ietf-lwig-cellular]

Arkkio, J., Eriksson, A., and A. Keranen, "Building Power-  
Efficient CoAP Devices for Cellular Networks", draft-ietf-  
lwig-cellular-00 (work in progress), August 2013.

[I-D.ietf-lwig-guidance]

Bormann, C., "Guidance for Light-Weight Implementations of  
the Internet Protocol Suite", draft-ietf-lwig-guidance-03  
(work in progress), February 2013.

[I-D.ietf-lwig-ikev2-minimal]

Kivinen, T., "Minimal IKEv2", draft-ietf-lwig-  
ikev2-minimal-01 (work in progress), October 2013.

[I-D.ietf-lwig-terminology]

Bormann, C., Ersue, M., and A. Keranen, "Terminology for  
Constrained Node Networks", draft-ietf-lwig-terminology-05  
(work in progress), July 2013.

[I-D.ietf-lwig-tls-minimal]

Kumar, S., Keoh, S., and H. Tschofenig, "A Hitchhiker's Guide to the (Datagram) Transport Layer Security Protocol for Smart Objects and Constrained Node Networks", draft-ietf-lwig-tls-minimal-00 (work in progress), September 2013.

[I-D.jennings-core-transitive-trust-enrollment]  
Jennings, C., "Transitive Trust Enrollment for Constrained Devices", draft-jennings-core-transitive-trust-enrollment-01 (work in progress), October 2012.

[I-D.kovatsch-lwig-class1-coap]  
Kovatsch, M., "Implementing CoAP for Class 1 Devices", draft-kovatsch-lwig-class1-coap-00 (work in progress), October 2012.

[I-D.kovatsch-lwig-coap]  
Kovatsch, M., Bergmann, O., Dijk, E., He, X., and C. Bormann, "CoAP Implementation Guidance", draft-kovatsch-lwig-coap-01 (work in progress), July 2013.

[I-D.li-core-coap-payload-length-option]  
Li, K., "CoAP Payload-Length Option Extension", draft-li-core-coap-payload-length-option-02 (work in progress), August 2013.

[I-D.pporamba-dtls-certkey]  
Porambage, P., Kumar, P., Gurtov, A., Ylianttila, M., and E. Harjula, "Certificate based keying scheme for DTLS secured IoT", draft-pporamba-dtls-certkey-00 (work in progress), June 2013.

[I-D.rahman-core-sleepy-problem-statement]  
Rahman, A., Fossati, T., Loreto, S., and M. Vial, "Sleepy Devices in CoAP - Problem Statement", draft-rahman-core-sleepy-problem-statement-01 (work in progress), October 2012.

[I-D.rahman-core-sleepy]  
Rahman, A., "Enhanced Sleepy Node Support for CoAP", draft-rahman-core-sleepy-04 (work in progress), October 2013.

[I-D.sarikaya-core-secure-bootsolution]  
Sarikaya, B., "Security Bootstrapping Solution for Resource-Constrained Devices", draft-sarikaya-core-secure-bootsolution-00 (work in progress), February 2013.

- [I-D.schmitt-two-way-authentication-for-iot]  
Schmitt, C., Stiller, B., Kothmayr, T., and W. Hu, "DTLS-based Security with two-way Authentication for IoT", draft-schmitt-two-way-authentication-for-iot-01 (work in progress), October 2013.
- [I-D.seitz-core-sec-usecases]  
Seitz, L., Gerdes, S., and G. Selander, "Use cases for CoRE security", draft-seitz-core-sec-usecases-00 (work in progress), September 2013.
- [I-D.selander-core-access-control]  
Selander, G., Sethi, M., and L. Seitz, "Access Control Framework for Constrained Environments", draft-selander-core-access-control-01 (work in progress), October 2013.
- [I-D.silverajan-core-coap-alternative-transports]  
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-03 (work in progress), October 2013.
- [I-D.urien-core-racs]  
Urien, P., "Remote APDU Call Secure (RACS)", draft-urien-core-racs-00 (work in progress), August 2013.
- [I-D.vanderstok-core-dna]  
Stok, P., Lynn, K., and A. Brandt, "CoRE Discovery, Naming, and Addressing", draft-vanderstok-core-dna-02 (work in progress), July 2012.
- [I-D.vial-core-mirror-proxy]  
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-proxy-01 (work in progress), July 2012.
- [I-D.vial-core-mirror-server]  
Vial, M., "CoRE Mirror Server", draft-vial-core-mirror-server-01 (work in progress), April 2013.
- [I-D.zhu-core-groupauth]  
Zhu, J. and M. Qi, "Group Authentication", draft-zhu-core-groupauth-01 (work in progress), September 2013.
- [REST]  
Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <[http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.



- [RFC2418] Bradner, S., "IETF Working Group Guidelines and Procedures", BCP 25, RFC 2418, September 1998.
- [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, June 1999.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC4815] Jonsson, L-E., Sandlund, K., Pelletier, G., and P. Kremer, "RObust Header Compression (ROHC): Corrections and Clarifications to RFC 3095", RFC 4815, February 2007.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.
- [SB] Bormann, C., Castellani, A., and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes", DOI 10.1109/MIC.2012.29, 2012.
- [WEI] Shelby, Z. and C. Bormann, "6LoWPAN: the Wireless Embedded Internet", ISBN 9780470747995, 2009.

#### Author's Address

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: cabo@tzi.org

Internet Engineering Task Force  
Internet-Draft  
Intended status: Informational  
Expires: January 17, 2013

Z. Cao  
China Mobile  
Y. Ma  
Hitachi R&D China  
H. Deng  
China Mobile  
R. Zhang  
China Telecom  
July 16, 2012

HTTP-COAP Proxy Discovery using Link-format  
draft-cao-core-pd-02

Abstract

This document discusses the problem of HTTP-COAP proxy discovery and proposes a method of using Link-format to do the job.

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 January 17, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|  |   |
|--|---|
| 1. Introduction . . . . .                  | 3 |
| 1.1. Requirements Language . . . . .       | 3 |
| 2. Scenario . . . . .                      | 3 |
| 3. Problem Formation . . . . .             | 3 |
| 4. Link-format Proxy Discovery . . . . .   | 4 |
| 5. Design Consideration . . . . .          | 5 |
| 6. Existing Discovery Mechanisms . . . . . | 5 |
| 7. Acknowledgements . . . . .              | 6 |
| 8. IANA Considerations . . . . .           | 6 |
| 9. Security Considerations . . . . .       | 6 |
| 10. References . . . . .                   | 6 |
| 10.1. Normative References . . . . .       | 6 |
| 10.2. Informative References . . . . .     | 7 |
| Authors' Addresses . . . . .               | 7 |

## 1. Introduction

CoAP [I-D.ietf-core-coap] is a RESTful protocol designed for constrained devices. The ultimate goal of CoAP is to enable the "Web of Things" concept, which connects the smart sensor network with the global internet. Although CoAP has been implemented on various platforms, the rest of web is still dominated by HTTP. As a result, it is desirable to interconnect the HTTP and CoAP via some intermediary proxy. For example, the CoAP sensor client in the constrained network can access and update resources on the HTTP server, and also the HTTP client on the web can access and/or update resources on the CoAP server.

There are already some works discussing how to map HTTP to CoAP and vice versa. The basic mapping between HTTP and CoAP is described in Section 8 of [I-D.ietf-core-coap] . Further details of implementing the proxy, internal procedures and design choices are described in [I-D.castellani-core-http-mapping] .

Static configuration of HTTP-CoAP proxies is a straightforward way for the client to access the server. However, in many situations, static configuration is not enough to meet the requirements. For example, if the HTTP client would like to access a certain type of resource (temperature or humidity in a certain location, etc.), it is required that the client would find an appropriate proxy to serve the content.

### 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 RFC 2119 [RFC2119].

## 2. Scenario

One example scenario of proxy discovery comes from the requirements that integrates the smart network with the mobile network. The sensors want to find a mobile proxy that can proxy the information to the web. The sensors are static, running CoAP as a client, and wants to report information to the SNS website. The "mobile M2M devices" are nomadic and can serve as the CoAP-HTTP proxy. The sensor wants to discover who is nearby and can shepherd message to the Web.

## 3. Problem Formation

We divide the problem into two separated parts. The first is how a

CoAP client discovers a proxy to access the HTTP server. For example, the CoAP sensors want to report or get some information to a Web server. In this case, the CoAP sensor only acts as a client. In static configuration, the CoAP client is configured via DHCP or RSRA. But in dynamic environment, a mechanism for dynamic configuration is desired. This document mainly discusses this aspect.

The other case is how a HTTP client discovers a proxy to access the CoAP server. For example, the HTTP client wants to access a certain type of information in the constrained network, and would discover the proxy to the exact constrained sensor. In this case, the HTTP Client only accesss the sensor indirectly. In this case, the HTTP Client only needs to know the address or the domain name of the proxy node, and the proxy forwards the requests to the sensor node according to the sub-domain information or the path included in the URI within the request. But in this case, we believe that the DNS-SD infrastructures are sufficient to handle this problem. For example, [I-D.vanderstok-core-bc] has described detailed considerations of a DNS-SD based proxy discovery method for Building Control use cases. So, in this document we will not talk about this direction.

#### 4. Link-format Proxy Discovery

Before the CoAP sensor makes use of the CoAP-HTTP proxy, it must know the location of the proxy. There can be multiple ways to discover the proxy's location, including both static and dynamic methods. DHCP is one way to do that, and documented in another document. This document describes one way to discover the proxy by the CoRE link format [I-D.ietf-core-link-format].

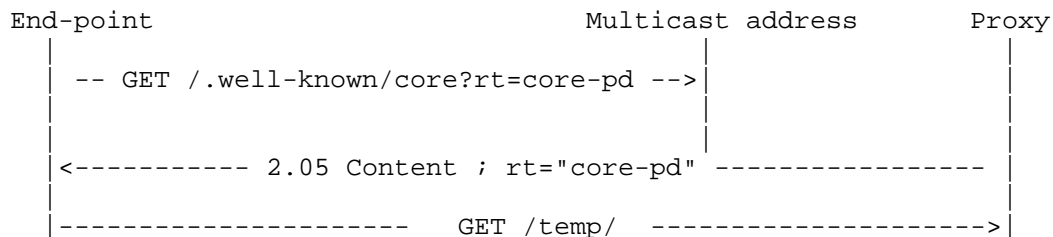
Note: Think of the way the user is configured with the http proxy in the enterprise network.

Discovery is performed by sending a multicast GET request to /.well-known/core and including a Resource Type (rt) parameter [I-D.ietf-core-link-format] with the value "core-pd" in the query string. Upon success, the response will contain a payload with a link format entry for each proxy discovered. The multicast IP address used will depend on the scope required and the multicast capabilities of the network. (If determined, IANA actions are required to assign a multicast address for this purpose)

The following example shows an end-point discover a locally available CoAP-HTTP proxy. The CoAP end-point sends a multicast GET request to the multicast address in the domain carrying a resource type "core-pd" indicating its discovery of a local proxy. Then the serving proxy responds the request with the rt="core-pd" and the

address of the proxy is carried within the Content payload. Afterwards, the CoAP sensor initiates the data-plane communication with the proxy directly.

To avoid the heavy load of multicast traffic on the link, there may be need of designate a ALL-COAP-MULTICAST address for proxy discovery.



Req: GET coap://[ff02::1]/.well-known/core?rt=core-pd

Res: 2.05 Content  
fe80::ff; rt="core-pd";

## 5. Design Consideration

There are some considerations with the above scheme. First, if all the nodes on the link is obliged to listen to the multicast message, the energy consumption would be high and unnecessary. To avoid all the nodes on the link receiving the GET message, we can use a "ALL-COAP" multicast address for such kind of request. Regarding the multicast addresses, there would be IANA actions on it. Second, the resource type (rt) definition of the proxy discovery should be defined by IANA.

## 6. Existing Discovery Mechanisms

There are many service discovery protocols, including:

1. DNS Service Discovery (DNS-SD) [I-D.cheshire-dnsext-dns-sd], part of Apple's Bonjour technology.
2. Service Location Protocol (SLP) [RFC2608]
3. Simple Service Discovery Protocol (SSDP) as used in Universal Plug and Play (UPnP)[upnp]

4. multicast DHCP (MDHCP) [mDHCP]
5. Web Proxy Autodiscovery Protocol (WPAD)[wpad]
6. Dynamic Host Configuration Protocol (DHCP) [RFC2131]

## 7. Acknowledgements

Some ideas in this document are according to the discussion between Zach Shelby on the problem. And authors also thank comments from Jari Arkko and Ralph Droms on IETF 82th meeting.

## 8. IANA Considerations

If the ideas in this document is determined by the working group, IANA actions are required to assign a multicast address for the purpose of HTTP-CoAP proxy discovery, as well as the link format for the proxy discovery.

## 9. Security Considerations

None.

## 10. References

### 10.1. Normative References

- [I-D.castellani-core-http-mapping]  
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-castellani-core-http-mapping-05 (work in progress), July 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-14 (work in progress), June 2012.
- [I-D.vanderstok-core-bc]

Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.

## 10.2. Informative References

- [I-D.cheshire-dnsext-dns-sd]  
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-11 (work in progress), December 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.
- [RFC2608] Guttman, E., Perkins, C., Veizades, J., and M. Day, "Service Location Protocol, Version 2", RFC 2608, June 1999.
- [RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.
- [mDHCP] "Multicast DHCP",  
<<http://technet.microsoft.com/en-us/library/cc958927.aspx>>.
- [upnp] "Universal Plug and Play", <<http://www.upnp.org/>>.
- [wpad] "Web Proxy Autodiscovery Protocol (WPAD)",  
<<http://tools.ietf.org/html/draft-ietf-wrec-wpad>>.

## Authors' Addresses

Zhen Cao  
China Mobile  
Xuanwumenxi Ave. No.32  
China, 100053  
China

Phone:  
Email: [zehn.cao@gmail.com](mailto:zehn.cao@gmail.com), [caozhen@chinamobile.com](mailto:caozhen@chinamobile.com)



Yuanchen Ma  
Hitachi R&D China

Phone:  
Fax:  
Email: ycma@hitachi.cn  
URI:

Hui Deng  
China Mobile

Phone:  
Fax:  
Email: denghui@chinamobile.com  
URI:

Rong Zhang  
China Telecom  
No.109 Zhongshandadao avenue  
Guangzhou, Tianhe 510630  
China

Phone:  
Fax:  
Email: zhangr@gsta.com  
URI:



CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 3, 2016

A. Castellani  
University of Padova  
S. Loreto  
Ericsson  
A. Rahman  
InterDigital Communications, LLC  
T. Fossati  
KoanLogic  
E. Dijk  
Philips Research  
July 2, 2015

Advanced Guidelines for HTTP-CoAP Mapping Implementations  
draft-castellani-core-advanced-http-mapping-06

Abstract

This draft describes advanced features for HTTP-CoAP proxy implementers. It details deployment options, discusses possible approaches for URI mapping, and provides useful considerations related to protocol translation.

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 January 3, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|   |    |
|---|----|
| 1. Terminology and Conventions . . . . .  | 3  |
| 2. Introduction . . . . .   | 3  |
| 3. Use Case: HTTP/IPv4-CoAP/IPv6 Proxy . . . . .  | 3  |
| 4. URI Mapping via HTTP Cache Control Extensions . . . . .                              | 6  |
| 5. Multiple Message Exchanges Mapping . . . . .   | 6  |
| 5.1. Relevant Features of Existing Standards . . . . .                                  | 6  |
| 5.1.1. Multipart Messages . . . . .   | 6  |
| 5.1.2. Immediate Message Delivery . . . . .   | 7  |
| 5.1.3. Detailing Source Information . . . . .   | 7  |
| 5.2. Multicast Mapping . . . . .  | 7  |
| 5.2.1. URI Identification and Mapping . . . . .   | 8  |
| 5.2.2. Request Handling . . . . .   | 8  |
| 5.2.3. Examples . . . . .   | 9  |
| 5.3. Multicast Response Caching . . . . .   | 11 |
| 5.4. Observe Mapping . . . . .  | 12 |
| 5.4.1. Identification . . . . .   | 12 |
| 5.4.2. Notification(s) Mapping . . . . .  | 14 |
| 5.4.3. Examples . . . . .   | 15 |
| 6. HTML5 Scheme Handler Registration . . . . .  | 21 |
| 7. Placement and Deployment . . . . .   | 21 |
| 8. Examples . . . . .   | 22 |
| 9. Acknowledgements . . . . .   | 24 |
| 10. IANA Considerations . . . . .   | 24 |
| 11. Security Considerations . . . . .   | 24 |
| 11.1. Cross-protocol Security Policy Mapping . . . . .                                  | 25 |
| 11.2. Subscription . . . . .  | 25 |
| 12. References . . . . .  | 25 |
| 12.1. Normative References . . . . .  | 25 |
| 12.2. Informative References . . . . .  | 26 |
| Appendix A. Internal Mapping Functions (from an Implementer's<br>Perspective) . . . . . | 27 |
| A.1. URL Map Algorithm . . . . .  | 28 |
| A.2. Security Policy Map Algorithm . . . . .  | 29 |
| A.3. Content-Type Map Algorithm . . . . .   | 30 |
| Authors' Addresses . . . . .  | 31 |

## 1. Terminology and Conventions

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 [RFC2119].

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

A device providing cross-protocol HTTP-CoAP mapping is called an HTTP-CoAP cross-protocol proxy (HC proxy).

At least two different kinds of HC proxies exist:

- o One-way cross-protocol proxy (1-way proxy): This proxy translates from a client of a protocol to a server of another protocol but not vice-versa.
- o Two-way (or bidirectional) cross-protocol proxy (2-way proxy): This proxy translates from a client of both protocols to a server supporting one protocol.

## 2. Introduction

RESTful protocols, such as HTTP [RFC2616] and CoAP [I-D.ietf-core-coap], can interoperate through an intermediary proxy which performs cross-protocol mapping.

A base reference for the mapping process is provided in [I-D.ietf-core-coap]. However, depending on the involved application, deployment scenario, or network topology, such mapping can be realized using a wide range of intermediaries.

Moreover, the process of implementing such a proxy can be complex, and details regarding its internal procedures and design choices deserve further discussion, which is provided in this document.

This draft itself is an evolution of the mapping features covered in [I-D.ietf-core-http-mapping].

## 3. Use Case: HTTP/IPv4-CoAP/IPv6 Proxy

This section covers the expected common use case regarding an HTTP/IPv4 client accessing a CoAP/IPv6 resource.

While HTTP and IPv4 are today widely adopted communication protocols in the Internet, a pervasive deployment of constrained nodes

exploiting the IPv6 address space is expected: enabling direct interoperability of such technologies is a valuable goal.

An HC proxy supporting IPv4/IPv6 mapping is said to be a v4/v6 proxy.

An HC v4/v6 proxy SHOULD always try to resolve the URI authority, and SHOULD prefer using the IPv6 resolution if available. The authority part of the URI is used internally by the HC proxy and SHOULD NOT be mapped to CoAP.

Figure 1 shows an HTTP client on IPv4 (C) accessing a CoAP server on IPv6 (S) through an HC proxy on IPv4/IPv6 (P). The DNS has an A record for "node.coap.something.net" resolving to the IPv4 address of the HC proxy, and an AAAA record with the IPv6 address of the CoAP server.

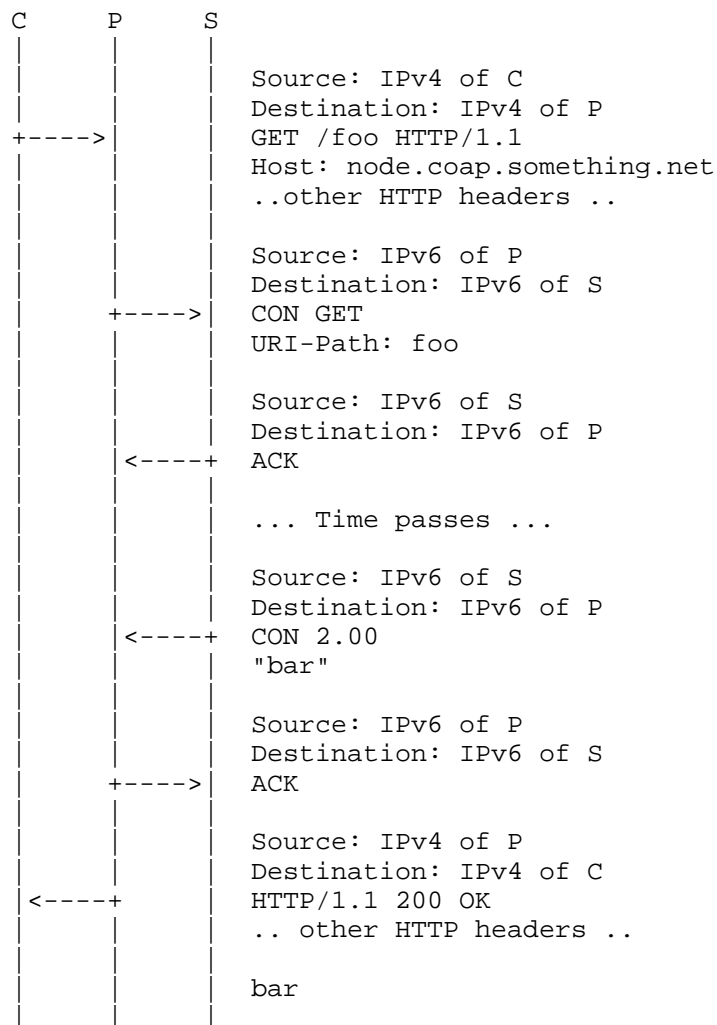


Figure 1: HTTP/IPv4 to CoAP/IPv6 Mapping

The proposed example shows the HC proxy operating also the mapping between IPv4 to IPv6 using the authority information available in any HTTP 1.1 request. This way, IPv6 connectivity is not required at the HTTP client when accessing a CoAP server over IPv6 only, which is a typical expected use case.

When P is an interception HC proxy, the CoAP request SHOULD have the IPv6 address of C as source (IPv4 can always be mapped into IPv6).

The described solution takes into account only the HTTP/IPv4 clients accessing CoAP/IPv6 servers; this solution does not provide a full fledged mapping from HTTP to CoAP.

In order to obtain a working deployment for HTTP/IPv6 clients, a different HC proxy access method may be required, or Internet AAAA records should not point to the node anymore (the HC proxy should use a different DNS database pointing to the node).

When an HC interception proxy deployment is used this solution is fully working even with HTTP/IPv6 clients.

#### 4. URI Mapping via HTTP Cache Control Extensions

An advanced strategy for triggering the cross-proxy that a translation is needed can be done via the HTTP Cache Control Extensions described in Section 5.2.3 of [RFC7234]. Specifically two new extensions can be defined, i.e. cross-coap and cross-coaps, that when included in a request to an HC forward cross-proxy translate the request to coap or coaps.

#### 5. Multiple Message Exchanges Mapping

This section discusses the mapping of the multicast and observe features of CoAP, which have no corresponding primitive in HTTP, and as such are not immediately translatable.

The mapping, which must be considered in both the arrow directions (H->C, C->H) may involve multi-part responses, as in the multicast use case, asynchronous delivery through HTTP bidirectional techniques, and HTTP Web Linking in order to reduce the semantics lost in the translation.

##### 5.1. Relevant Features of Existing Standards

Various features provided by existing standards are useful to efficiently represent sessions involving multiple messages.

##### 5.1.1. Multipart Messages

In particular, the "multipart/\*" media type, defined in Section 5.1 of [RFC2046], is a suitable solution to deliver multiple CoAP responses within a single HTTP payload. Each part of a multipart entity SHOULD be represented using "message/http" media type containing the full mapping of a single CoAP response as previously described.



### 5.1.2. Immediate Message Delivery

An HC proxy may prefer to transfer each CoAP response immediately after its reception. This is possible thanks to the HTTP Transfer-Encoding "chunked", that enables transferring single responses without any further delay.

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [RFC6202]. Large delays between chunks can lead the HTTP session to timeout, more details on this issue can be found in [I-D.thomson-hybi-http-timeout].

An HC proxy MAY prefer (e.g. to avoid buffering) to transfer each response related to a multicast request as soon as it comes in from the server. One possible way to achieve this result is using the "chunked" Transfer-Encoding in the HTTP response, to push individual responses until some trigger is fired (timeout, max number of messages, etc.).

An example showing immediate delivery of CoAP responses using HTTP chunks will be provided in Section 5.4, while describing its application to an observe session.

### 5.1.3. Detailing Source Information

Under some circumstances, responses may come from different sources (i.e. responses to a multicast request); in this case details about the actual source of each CoAP response MAY be provided to the client. Source information can be represented using HTTP Web Linking as defined in [RFC5988], by adding the actual source URI into each response using Link option with "via" relation type.

## 5.2. Multicast Mapping

In order to establish a multicast communication such a feature should be offered either by the network (i.e. IP multicast, link-layer multicast, etc.) or by a gateway (i.e. the HC proxy). Rationale on the methods available to obtain such a feature is out-of-scope of this document, and extensive discussion of group communication techniques is available in [I-D.ietf-core-groupcomm].

Additional considerations related to handling multicast requests mapping are detailed in the following sections.

### 5.2.1. URI Identification and Mapping

In order to successfully handle a multicast request, the HC proxy MUST successfully perform the following tasks on the URI:

Identification: The HC proxy MUST understand whether the requested URI identifies a group of nodes.

Mapping: The HC proxy MUST know how to distribute the multicast request to involved servers; this process is specific of the group communication technology used.

When using IPv6 multicast paired with DNS, the mapping to IPv6 multicast is simply done using DNS resolution. If the group management is performed at the proxy, the URI or part of it (i.e. the authority) can be mapped using some static or dynamic table available at the HC proxy. In Section 3.5 of [I-D.ietf-core-groupcomm] discusses a method to build and maintain a local table of multicast authorities.

### 5.2.2. Request Handling

When the HC proxy receives a request to a URI that has been successfully identified and mapped to a group of nodes, it SHOULD start a multicast proxying operation, if supported by the proxy.

Multicast request handling consists of the following steps:

Multicast TX: The HC proxy sends out the request on the CoAP side by using the methods offered by the specific group communication technology used in the constrained network;

Collecting RXs: The HC proxy collects every response related to the request;

Timeout: The HC proxy has to pay special attention in multicast timing, detailed discussion about timing depends upon the particular group communication technology used;

Distributing RXs to the client: The HC proxy can distribute the responses in two different ways: batch delivering them at the end of the process or on timeout, or immediately delivering them as they are available. Batch requires more caching and introduces delays but may lead to lower TCP overhead and simpler processing. Immediate delivery is the converse. A trade-off solution of partial batch delivery may also be feasible and efficient in some circumstances.

### 5.2.3. Examples

Figure 2 shows an HTTP client (C) requesting the resource `/foo` to a group of CoAP servers (S1/S2/S3) through an HC proxy (P) which uses IP multicast to send the corresponding CoAP request.

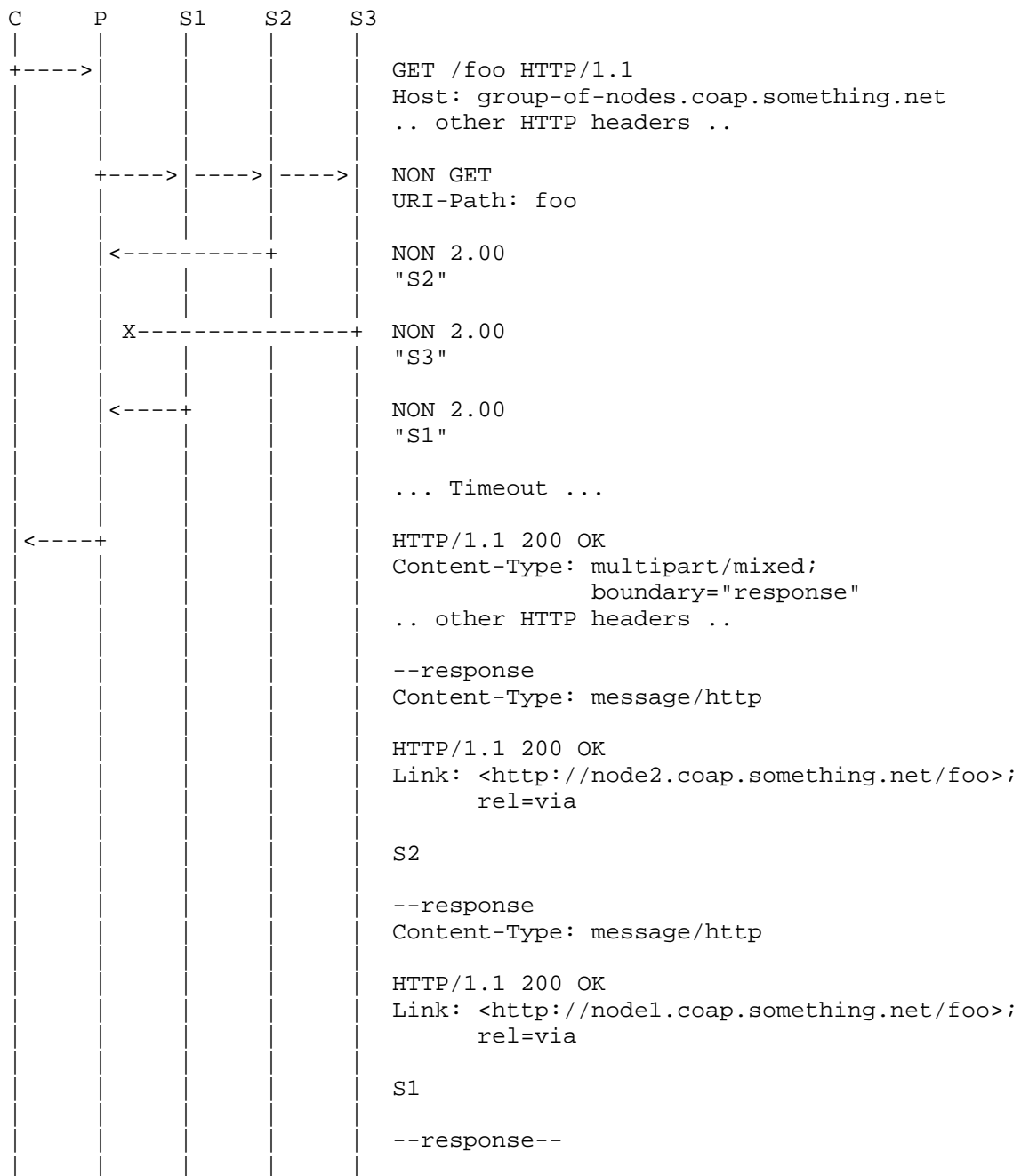


Figure 2: Unicast HTTP to Multicast CoAP Mapping

The example proposed in the above diagram does not make any assumption on which underlying group communication technology is available in the constrained network. Some detailed discussion is provided about it along the following lines.

C makes a GET request to `group-of-nodes.coap.something.net`. This domain name MAY either resolve to the address of P, or to the IPv6 multicast address of the nodes (if IP multicast is supported and P is an interception proxy), or the proxy P is specifically known by the client that sends this request to it.

To successfully start multicast proxying operation, the HC proxy MUST know that the destination URI involves a group of CoAP servers, e.g. the authority `group-of-nodes.coap.something.net` is known to identify a group of nodes either by using an internal lookup table, using DNS paired with IPv6 multicast, or by using some other special technique.

A specific implementation option is proposed to further explain the proposed example. Assume that DNS is configured such that all subdomain queries to `coap.something.net`, such as `group-of-nodes.coap.something.net`, resolve to the address of P. P performs the HC URI mapping by removing the 'coap' subdomain from the authority and by switching the scheme from 'http' to 'coap' (result: `"coap://group-of-node.something.net/foo"`); `"group-of-nodes.something.net"` is resolved to an IPv6 multicast address to which S1, S2 and S3 belong. The proxy handles this request as multicast and sends the request `"GET /foo"` to the multicast group .

### 5.3. Multicast Response Caching

We call perfect caching when the proxy uses only the cached representations to provide a response to the HTTP client. In the case of a multicast CoAP request, perfect caching is not adequate. This section updates the general caching and congestion control guidelines of with specific guidelines for the multicast use case.

Due to the inherent unreliable nature of the NON messages involved and since nodes may have dynamic membership in multicast groups, responding only with previously cached responses without issuing a new multicast request is not recommended. This perfect caching behaviour leads to miss responses of nodes that later joined the multicast group, and/or to repeatedly serve partial representations due to message losses. Therefore a multicast CoAP request SHOULD be sent by a HC proxy for each incoming request addressed to a multicast group.

Caching of multicast responses is still a valuable goal to pursue reduce network congestion, battery consumption and response latency.

Some considerations to be performed when adopting a multicast caching behaviour are outlined in the following paragraph.

Caching of multicast GET responses MAY be implemented by adopting some technique that takes into account either knowledge about dynamic characteristics of group membership (occurrence or frequency of group changes) or even better its full knowledge (list of nodes currently part of the group).

When using a technique exploiting this knowledge, valid cached responses SHOULD be served from cache.

#### 5.4. Observe Mapping

By design, and certainly not without a good rationale, HTTP lacks a publish-subscriber facility. This implies that the mapping of the CoAP observe semantics has to be created ad hoc, perhaps by making use of one of the well-known HTTP techniques currently employed to establish an HTTP bidirectional connection with the target resource - as documented in [RFC6202].

In the following sections we will describe some of the approaches that can be used to identify an observable resource and to create the communication bridging needed to set up an end to end HTTP-CoAP observation.

##### 5.4.1. Identification

In order to appropriately process an observe request, the HC proxy needs to know whether a given request is intended to establish an observation on the target resource, instead of triggering a regular request-response exchange.

At least two different approaches to identify such special requests exist, as discussed below.

##### 5.4.1.1. Observable URI Mapping

An URI is said to be observable whenever every request to it implicitly requires the establishment of an HTTP bidirectional connection to the resource.

Such subscription to the resource is always paired, if possible, to a CoAP observe session to the actual resource being observed. In general, multiple connections that are active with a single observable resource at the same time, are multiplexed to the single observe session opened by the intermediary. Its notifications are then de-multiplexed by the HC proxy to every HTTP subscriber.

An intermediary MAY pair a couple of distinct HTTP URIs to a single CoAP observable resource: one providing the usual request-response mediated access to the resource, and the other that always triggers a CoAP observe session.

#### 5.4.1.1.1. Discovery

As shown in Figure 3, in order to know whether an URI is observable, an HTTP UA MAY do a pre-flight request to the target resource using the HTTP OPTIONS method (see section 6.2 of [I-D.ietf-httpbis-p2-semantics]) to discover the communication options available for that resource.

If the resource supports observation, the proxy adds a Link Header [RFC5988] with the "obs" attribute as link-param (see Section 7 of [I-D.ietf-core-observe]).

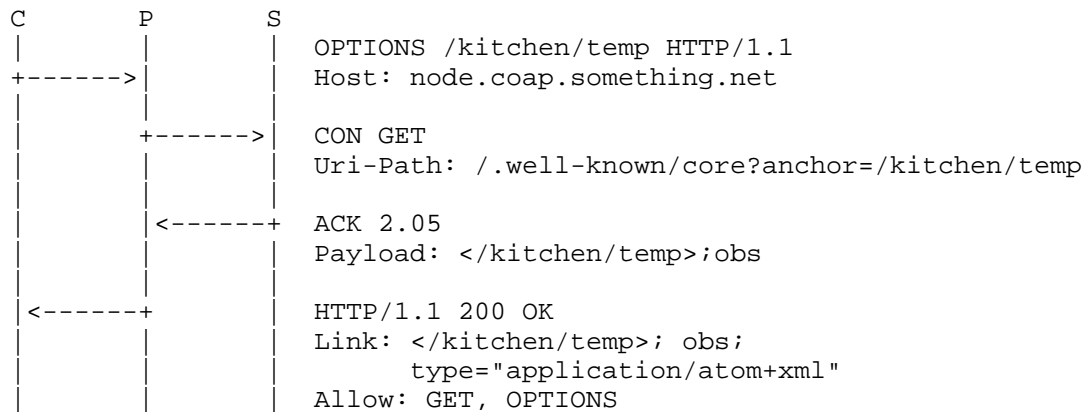


Figure 3: Discover Observability with HTTP OPTIONS

#### 5.4.1.2. Differentiation Using HTTP Header

Discerning an observation request through in-protocol means, e.g. via the presence and values of some HTTP metadata, avoids introducing static "observable" URIs in the HC proxy namespace. Though ideally the former should be preferred, there seems to be no standard way to use one of the established HTTP headers to convey the observe semantics.

Standardizing such methods is out-of-scope of this document, so we just point out some possible approaches that in the future may be used to differentiate observation requests from regular requests.

#### 5.4.1.2.1. Expect Header

The first method involves the use of the Expect header as defined in Section 9.3 of [I-D.ietf-httpbis-p2-semantics]. Whenever an HC proxy receives a request with a "206-partial-content" expectation, the proxy MUST fulfill this expectation by pairing this request to either a new or existing observe session to the resource.

If the proxy is unable to observe the resource, or if the observation establishment fails, the proxy MUST reply to the client with "417 Expectation Failed" status code.

Given that the Expect header is processed hop-by-hop, this method will fail immediately in case a proxy not supporting this expectation is traversed. For this reason, at present, the said approach can't be used in the public Internet.

#### 5.4.1.2.2. Prefer Header

A second, very similar, approach involves the use of the Prefer header, defined in [I-D.snell-http-prefer]. The HTTP user agent expresses the preference to establish an observation with the target resource by including a "streaming" preference to request an HTTP Streaming session, or a "long-polling" preference to signal to the proxy its intended polling behaviour (see [RFC6202]).

A compliant HC proxy will try to fulfill the preference, and manifest observation establishment success by responding with a status code of "206 Partial Content". The observation request fails, falling back to a single response, whenever the status code is different from 206.

This approach will never fail immediately, differently from the previous one, even across a chain of unaware proxies; however, as documented in [RFC6202], caching intermediaries may interfere, delay or block the HTTP bidirectional connection, making this approach unacceptable when no weak consistency of the resource can be tolerated by the requesting UA.

#### 5.4.2. Notification(s) Mapping

Multiplexing notifications using a single HTTP bidirectional session needs some further considerations about the selection of the media type that best fits this specific use case.

The usage of two different content-types that are suitable for carrying multiple notifications in a single session, is discussed in the following sections.



#### 5.4.2.1. Multipart Messaging

As already discussed in Section 5.1.1 for multicasting, the "multipart/\*" media type is a suitable solution to deliver multiple CoAP notifications within a single HTTP payload.

As in the multicast case, each part of the multipart entity MAY be represented using a "message/http" media type, containing the full mapping of the single CoAP notification mapped, so that CoAP envelope information are preserved (e.g. the response code).

A more sophisticated mapping could use multipart/mixed with native or translated media type.

#### 5.4.2.2. Using ATOM Feeds

Popular observable resources with refresh rates higher than a couple of seconds may be treated as Atom feeds [RFC4287], especially with delay tolerant user agents and where persistence is required.

Figure 3 shows a resource supporting 'application/atom+xml' media-type. In such case clients can listen to update notification by regularly polling the resource via opportunely spaced GETs, i.e. driven by the advertised max-age value.

#### 5.4.3. Examples

Figure 4 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

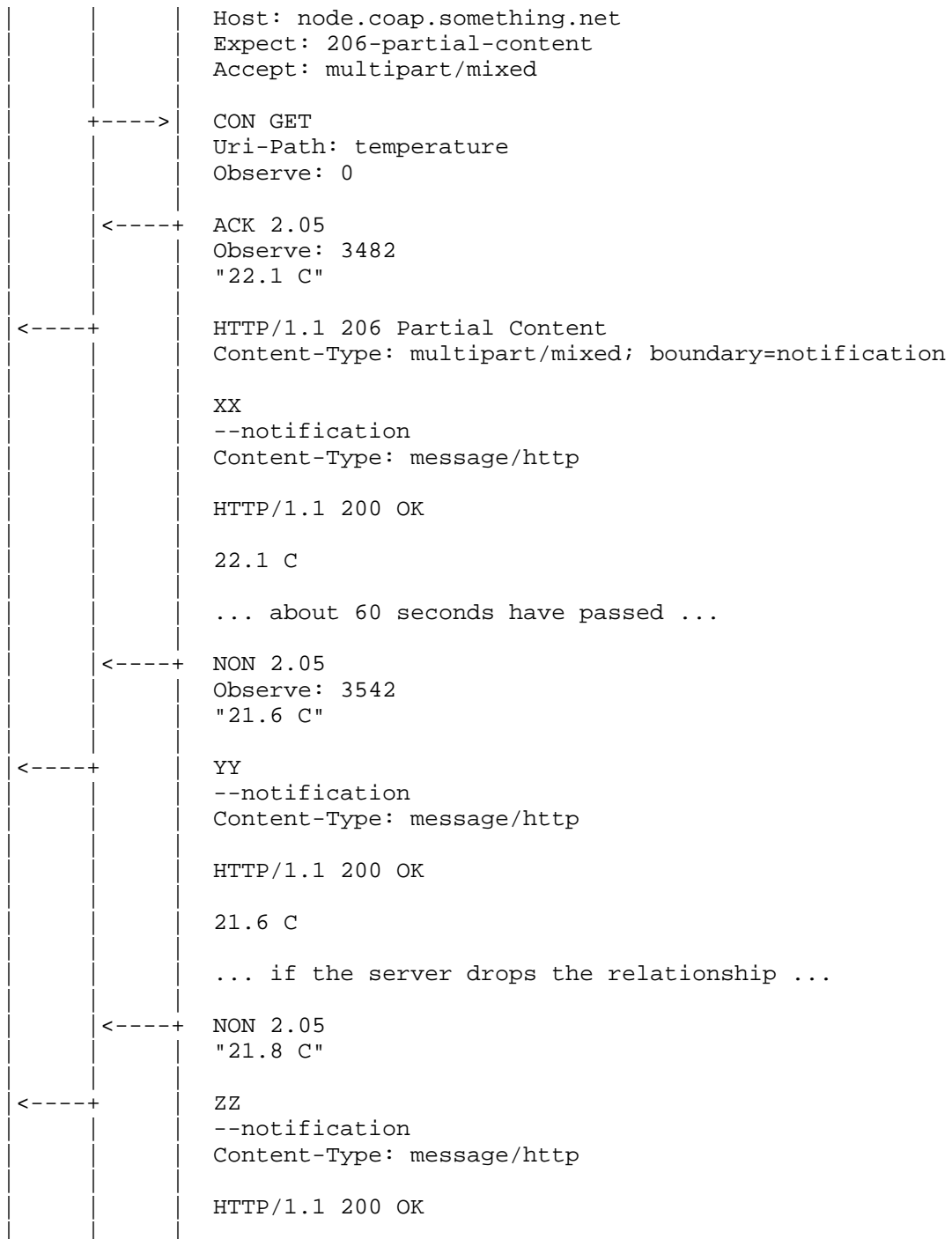
C manifests its intention to observe T by including the Expect Header in the request; if P or S do not support this interaction, the request MUST fail with "417 Expectation Failed" return code. In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" return code.

At every notification corresponds the emission of a HTTP chunk containing a single part, which contains a "message/http" payload containing the full mapping of the notification. When the observation is dropped by the CoAP server, the HTTP streaming session is closed.

```

C      P      S
|      |      |
+---->|      | GET /temperature HTTP/1.1

```



|  |  |  |                  |
|--|--|--|------------------|
|  |  |  | 21.8 C           |
|  |  |  | --notification-- |
|  |  |  | 0                |

Figure 4: HTTP Streaming to CoAP Observe

Figure 5 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "temperature" (T) available on S.

C manifests its intention to observe T by including the Prefer Header in the request; if P or S do not support this interaction, the request silently fails if a status code "200 OK" is returned, which means that no further notification is expected on that session.

In the presented example, both P and C support this interaction, and the subscription is successful, as stated by the "206 Partial Content" status code. At every notification a new response is sent to the pending client, always containing the "206 Partial Content" status code, to indicate that the observe session is still active, so that C can issue a new long-polling request immediately after this notification.

If the observation relationship is dropped by S, P notifies the last received content using the "200 OK" status code, indicating that no further notification is expected on this observe session.

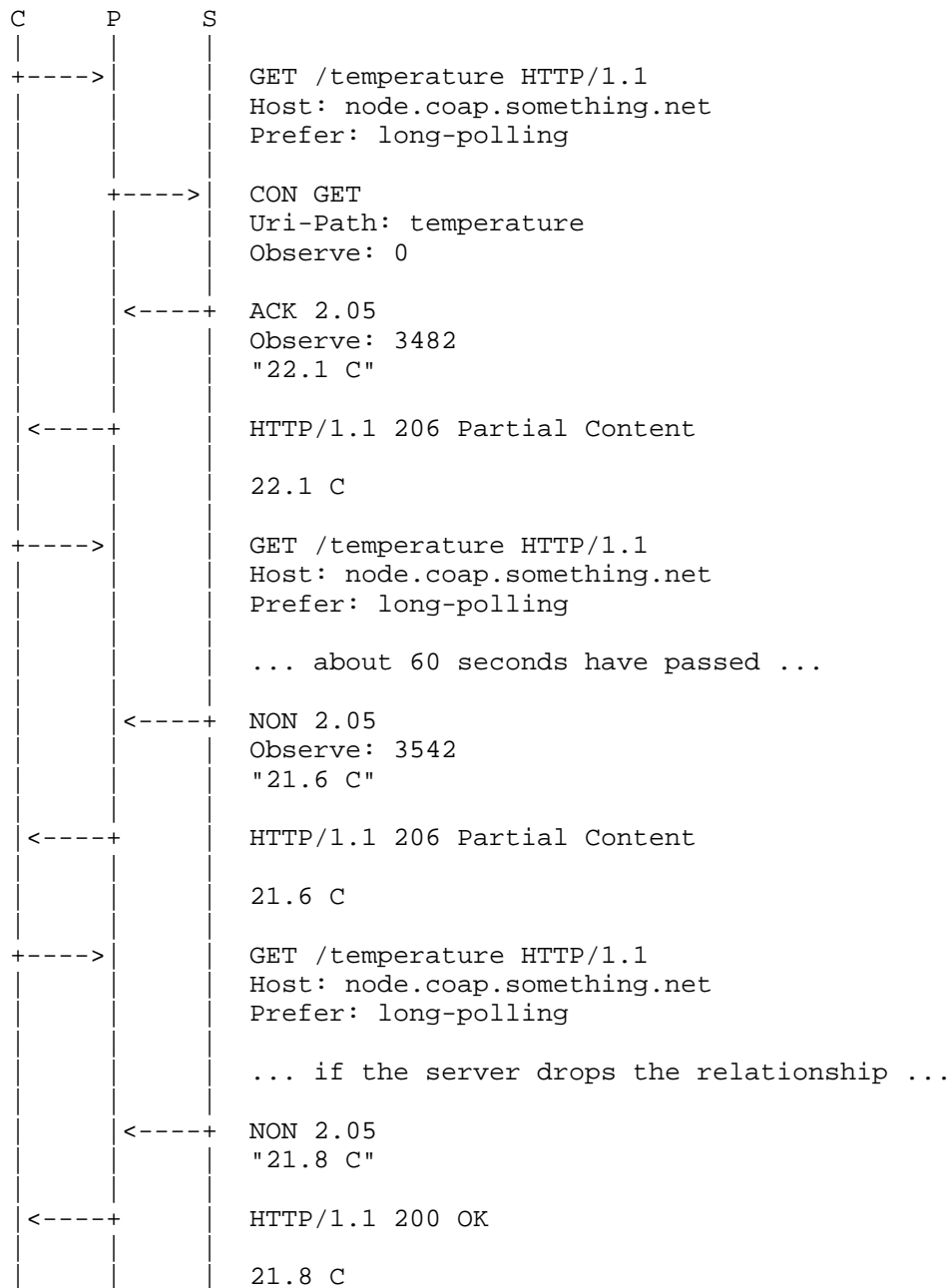


Figure 5: HTTP Long Polling to CoAP Observe

Figure 6 shows the interaction between an HTTP client (C), an HC proxy (P), and a CoAP server (S) for the observation of the resource "kitchen/temp" (T) available on S.

It is assumed that the HC proxy knows that the requested resource is observable (since perhaps being asked beforehand to discover its properties as described in Figure 3.) When asked by the HTTP client to retrieve the resource, it requests an observation - in case it weren't already in place - and then sends the collected data to the client as an Atom feed. The data coming through in the constrained network is stored locally on the proxy, and forwarded when further requests are received on the HTTP side. As already said, using the Atom format has two main advantages: first, there is always a "current" feed, but there may also be a complete log made available to HTTP clients; secondly, the HTTP intermediaries can play a substantial role in absorbing a fair amount of the load on the HC proxy. The latter is a very important property when the requested resource is or becomes very popular.

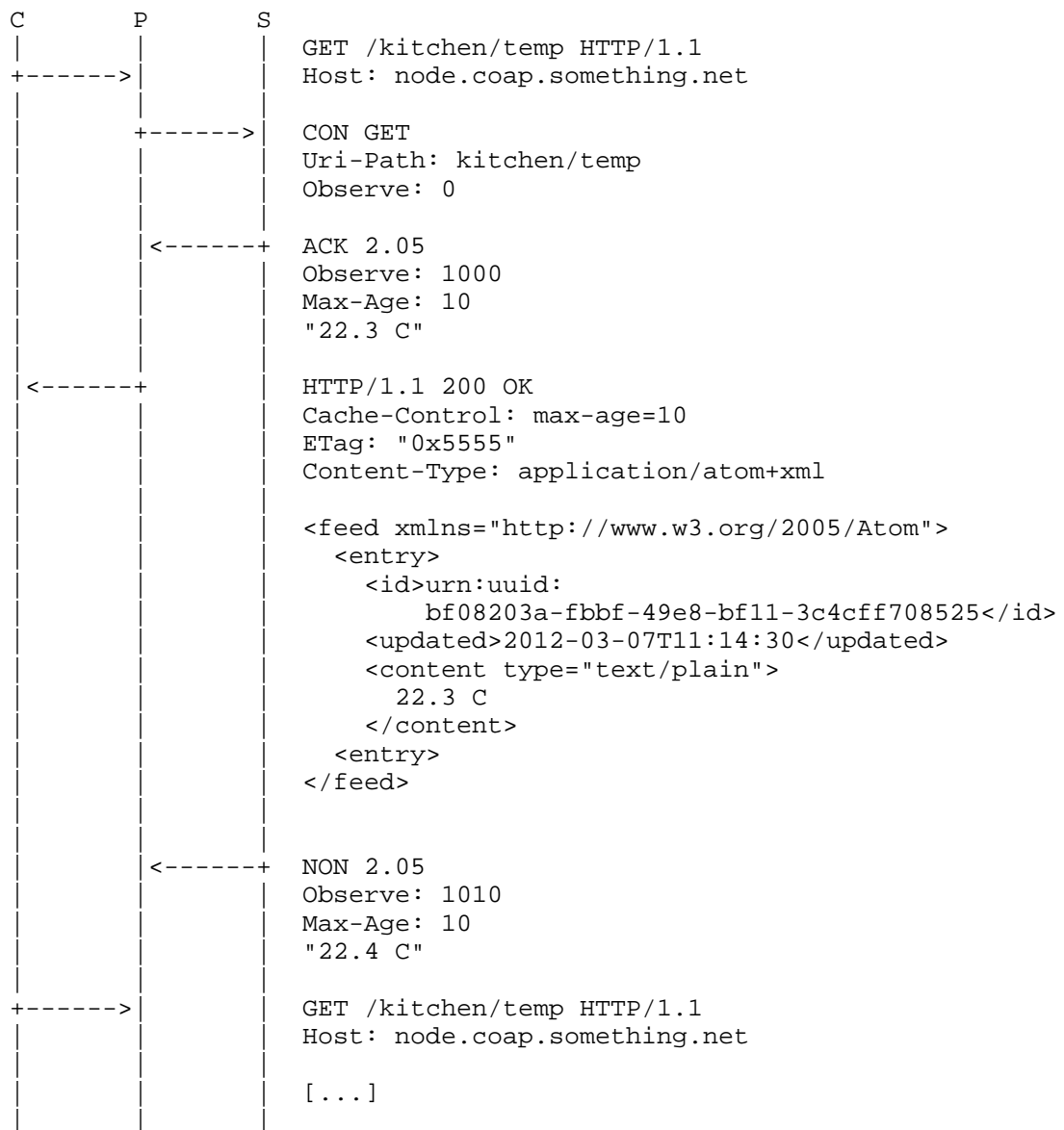


Figure 6: Observation via Atom feeds

## 6. HTML5 Scheme Handler Registration

The draft HTML5 standard offers a mechanism that allows an HTTP user agent to register a custom scheme handler through an HTML5 web page. This feature permits to an HC proxy to be registered as "handler" for URIs with the 'web+coap' or 'web+coaps' schemes using an HTML5 web page which embeds the custom scheme handler registration call `registerProtocolHandler()` described in Section 6.5.1.2 of [W3C.HTML5].

Example: the HTML5 homepage of a HC proxy at `h2c.example.org` could include the method call:

```
registerProtocolHandler('web+coap','proxy?url=%s','example HC proxy')
```

This registration call will prompt the HTTP user agent to ask for the user's permission to register the HC proxy as a handler for all 'web+coap' URIs. If the user accepts, whenever a 'web+coap' link is requested, the request will be fulfilled through the HC proxy: URI `"web+coap://foo.org/a"` will be transformed into URI `"http://h2c.example.org/proxy?url=web+coap://foo.org/a"`.

## 7. Placement and Deployment

In typical scenarios, for communication from a CoAP client to an HTTP origin server, the HC proxy is expected to be located on the client-side (CS). Specifically, the HC proxy is expected to be deployed at the edge of the constrained network as shown in Figure 7.

The arguments supporting CS placement are as follows:

Client/Proxy/Network configuration overhead: CoAP clients require either static proxy configuration or proxy discovery support. This overhead is simplified if the proxy is placed on the same network domain of the client.

TCP/UDP: Translation between CoAP and HTTP requires also UDP to TCP mapping; UDP performance over the unconstrained Internet may not be adequate. In order to minimize the number of required retransmissions on the constrained part of the network and the overall reliability, TCP/UDP conversion SHOULD be performed as soon as possible in the network path.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, thus a CS placement, collecting all the traffic, is strategic for this need.

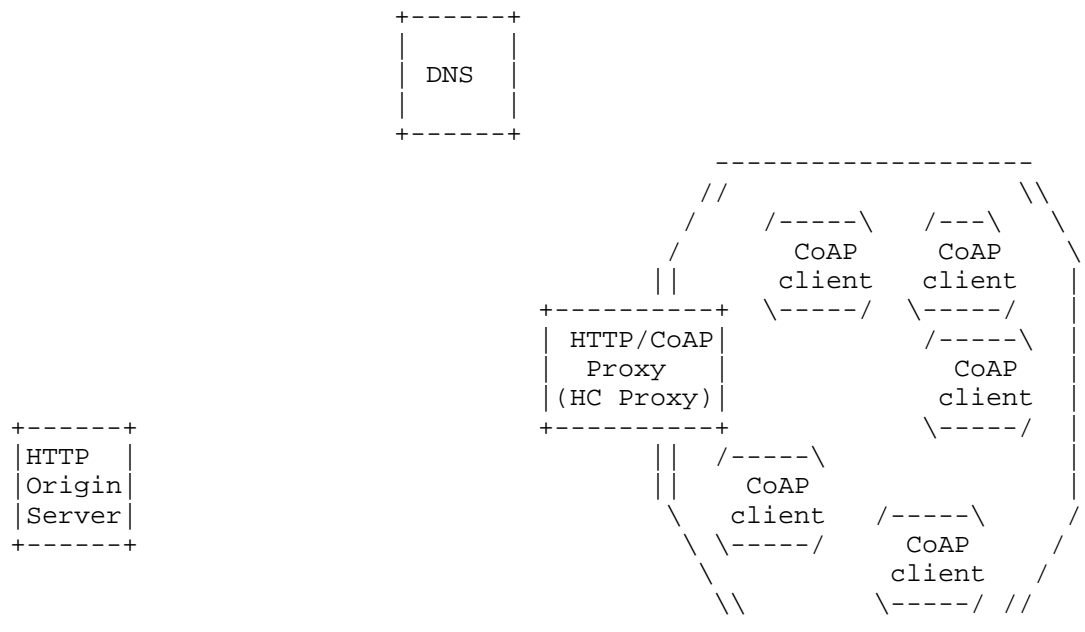


Figure 7: Client-side HC Proxy Deployment Scenario

## 8. Examples

Figure 8 shows an example implementation of a basic CoAP GET request with an HTTP URI as the value of a Proxy-URI option. The proxy retrieves a representation of the target resource from the HTTP origin server. It converts the payload to a UTF-8 charset, calculates the Max-Age Option from the Expires header field, and derives an entity-tag from the ETag header field.



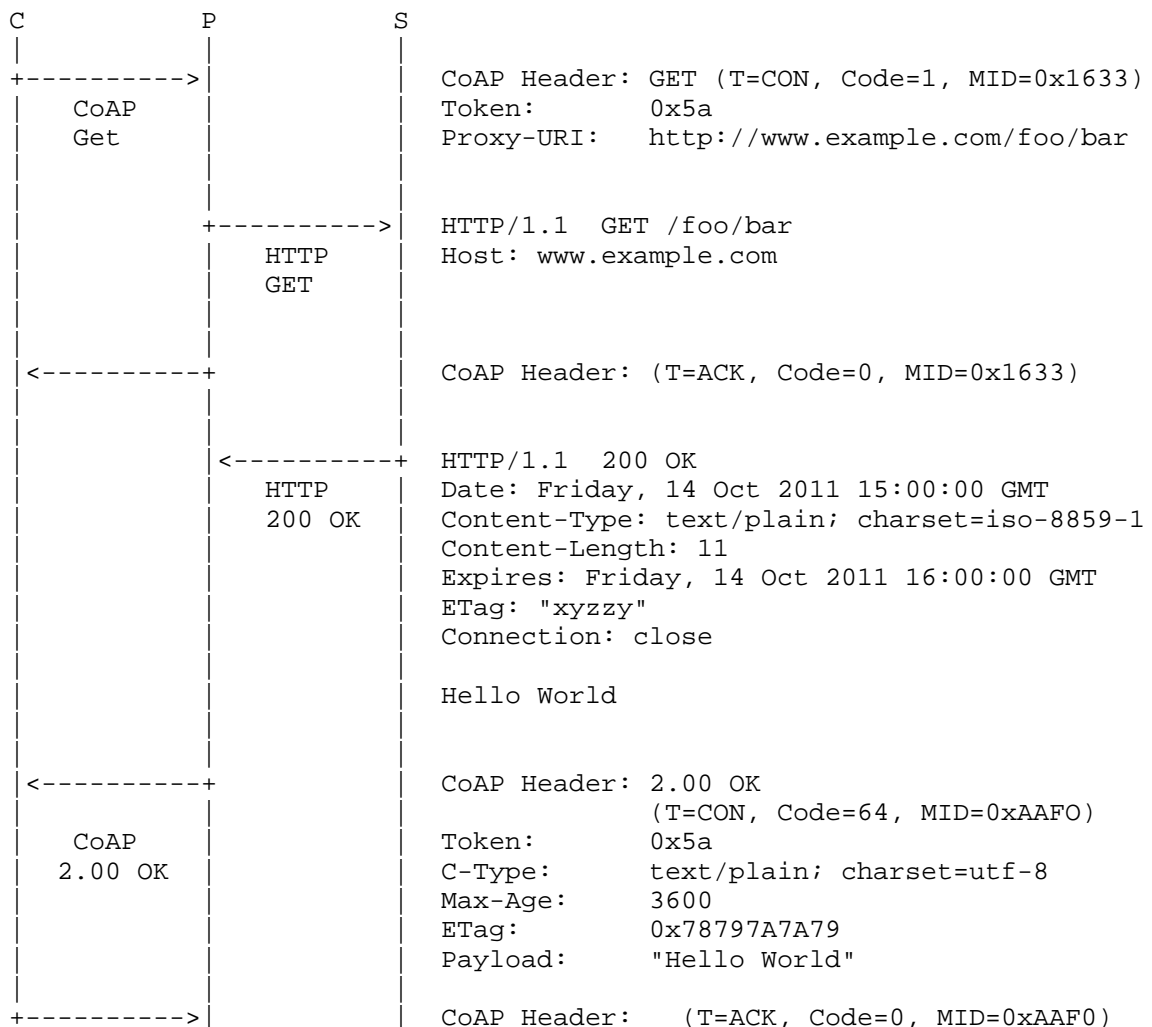


Figure 8: A Basic CoAP-HTTP GET Request

The example in Figure 9 builds on the previous example and shows an implementation of a GET request that includes a previously returned ETag Option. The proxy makes a Conditional Request to the HTTP origin server by including an If-None-Match header field in the HTTP GET Request. The CoAP response indicates that the response stored by the client is fresh. It includes a Max-Age Option calculated from the HTTP response's Expires header field.

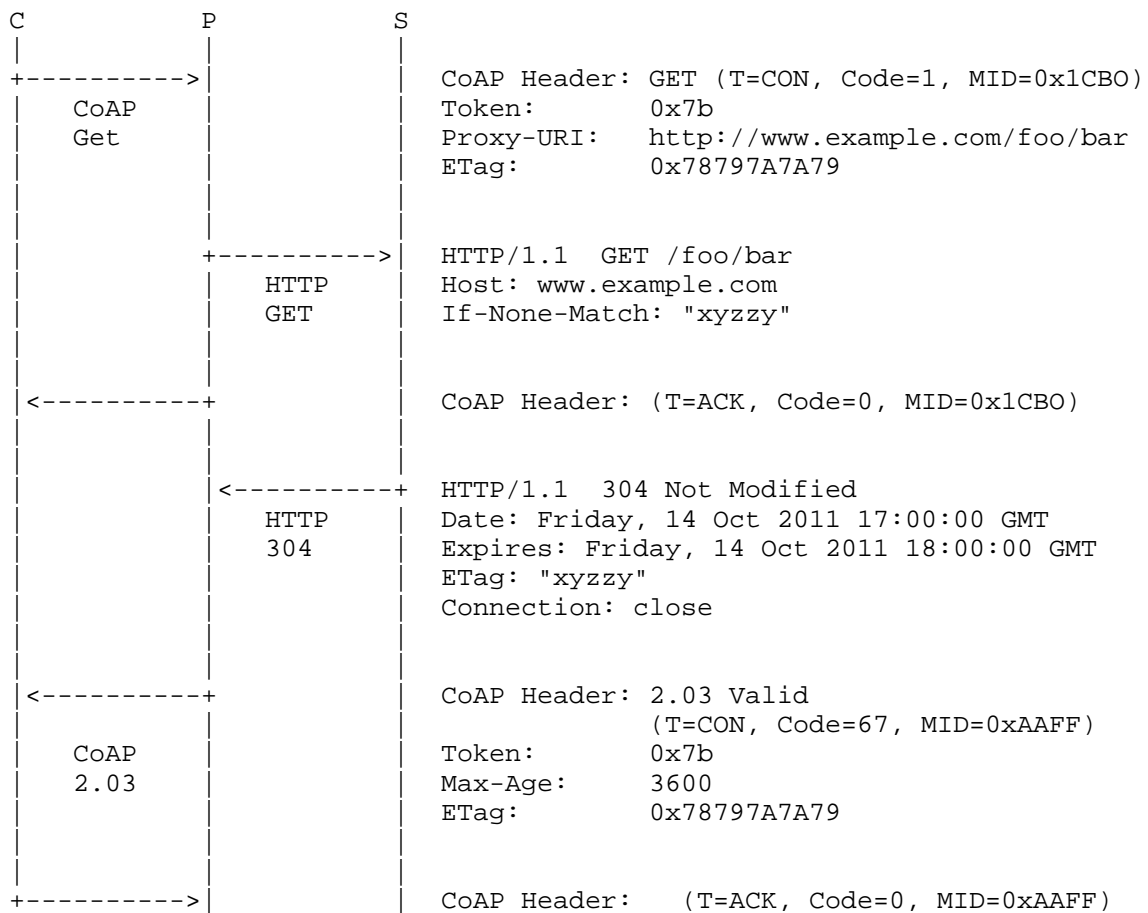


Figure 9: A CoAP-HTTP GET Request with an ETag Option

## 9. Acknowledgements

TBD.

## 10. IANA Considerations

This memo includes no request to IANA.

## 11. Security Considerations

### 11.1. Cross-protocol Security Policy Mapping

At the moment of this writing, CoAP and HTTP are missing any cross-protocol security policy mapping.

The HC proxy SHOULD flexibly support security policies between the two protocols, possibly as part of the HC URI mapping function, in order to statically map HTTP and CoAP security policies at the proxy (see Appendix A.2 for an example.)

### 11.2. Subscription

As noted in Section 7 of [I-D.ietf-core-observe], when using the observe pattern, an attacker could easily impose resource exhaustion on a naive server who's indiscriminately accepting observer relationships establishment from clients. The converse of this problem is also present, a malicious client may also target the HC proxy itself, by trying to exhaust the HTTP connection limit of the proxy by opening multiple subscriptions to some CoAP resource.

Effective strategies to reduce success of such a DoS on the HTTP side (by forcing prior identification of the HTTP client via usual web authentication mechanisms), must always be weighted against an acceptable level of usability of the exposed CoAP resources.

## 12. References

### 12.1. Normative References

- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-12 (work in progress), June 2013.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [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-http-mapping]  
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best Practices for HTTP-CoAP Mapping Implementation", draft-ietf-core-http-mapping-00 (work in progress), June 2013.

- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.
- [I-D.ietf-httpbis-p1-messaging]  
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", draft-ietf-httpbis-p1-messaging-22 (work in progress), February 2013.
- [I-D.ietf-httpbis-p2-semantics]  
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", draft-ietf-httpbis-p2-semantics-22 (work in progress), February 2013.
- [I-D.thomson-hybi-http-timeout]  
Thomson, M., Loreto, S., and G. Wilkins, "Hypertext Transfer Protocol (HTTP) Keep-Alive Header", draft-thomson-hybi-http-timeout-03 (work in progress), July 2012.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

## 12.2. Informative References

- [I-D.bormann-core-simple-server-discovery]  
Bormann, C., "CoRE Simple Server Discovery", draft-bormann-core-simple-server-discovery-01 (work in progress), March 2012.

- [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.snell-http-prefer]  
Snell, J., "Prefer Header for HTTP", draft-snell-http-prefer-18 (work in progress), January 2013.
- [I-D.vanderstok-core-bc]  
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.
- [RFC7234] Fielding, R., Nottingham, M., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014.
- [W3C.HTML5]  
Hickson, I., "HTML5", World Wide Web Consortium WD (work in progress) WD-html5-20111018, October 2011, <<http://dev.w3.org/html5/spec/>>.

#### Appendix A. Internal Mapping Functions (from an Implementer's Perspective)

At least three mapping functions have been identified, which take place at different stages of the HC proxy processing chain, involving the URL, Content-Type and Security Policy translation.

All these maps are required to have at least URL granularity so that, in principle, each and every requested URL may be treated as an independent mapping source.

In the following, the said map functions are characterized via their expected input and output, and a simple, yet sufficiently rich, configuration syntax is suggested.

In the spirit of a document providing implementation guidance, the specification of a map grammar aims at putting the basis for a reusable software component (e.g. a stand-alone C library) that many different proxy implementations can link to, and benefit from.

#### A.1. URL Map Algorithm

In case the HC proxy is a reverse proxy, i.e. it acts as the origin server in face of the served network, the URL of the resource requested by its clients (perhaps having an 'http' scheme) shall be mapped to the real resource origin (perhaps in the 'coap' scheme).

In case HC is a forward proxy, no URL translation is needed since the client already knows the "real name" of the resource.

An interception HC proxy, instead, MAY use the homogeneous mapping strategy to operate without any pre-configuration need.

As noted in Appendix B of [RFC3986] any correctly formatted URL can be matched by a POSIX regular expression. By leveraging on this property, we suggest a syntax that describes the URL mapping in terms of substituting the regex-matching portions of the requested URL into the mapped URL template.

E.g.: given the source regular expression `^http://example.com/coap/.*$` and destination template `coap://$1` (where `$1` stands for the first - and only in this specific case - substring matched by the regex pattern in the source), the input URL `"http://example.com/coap/node1/resource2"` translates to `"coap://node1/resource2"`.

This is a well established technique used in many today's web components (e.g. Django URL dispatcher, Apache `mod_rewrite`, etc.), which provides a compact and powerful engine to implement what essentially is an URL rewrite function.

INPUT  
\* requested URL

OUTPUT  
\* target URL

SYNTAX  
url\_map [rule name] {  
 requested\_url <regex>  
 mapped\_url <regex match subst template>  
}

EXAMPLE 1  
url\_map homogeneous {  
 requested\_url '^http://.\*\$'  
 mapped\_url 'coap//\$1'  
}

EXAMPLE 2  
url\_map embedded {  
 requested\_url '^http://example.com/coap/.\*\$'  
 mapped\_url 'coap//\$1'  
}

Note that many different url\_map records may be given in order to build the whole mapping function. Each of these records can be queried (in some predefined order) by the HC proxy until a match is found, or the list is exhausted. In the latter case, depending on the mapping policy (only internal, internal then external, etc.) the original request can be refused, or the same mapping query is forwarded to one or more external URL mapping components.

## A.2. Security Policy Map Algorithm

In case the "incoming" URL has been successfully translated, the HC proxy must lookup the security policy, if any, that needs to be applied to the request/response transaction carried on the "outgoing" leg.

## INPUT

- \* target URL (after URL map has been applied)
- \* original requester identity (given by cookie, or IP address, or crypto credentials/security context, etc.)

## OUTPUT

- \* security context that will be applied to access the target URL

## SYNTAX

```

sec_map [rule name] {
    target_url      <regex>          -- one or more
    requester_id    <TBD>
    sec_context     <TBD>
}

```

## EXAMPLE

```
<TBD>
```

## A.3. Content-Type Map Algorithm

In case a set of destination URLs is known as being limited in handling a narrow subset of mime types, a content-type map can be configured in order to let the HC proxy transparently handle the compatible/lossless format translation.

## INPUT

- \* destination URL (after URL map has been applied)
- \* original content-type

## OUTPUT

- \* mapped content-type

## SYNTAX

```

ct_map {
    target_url <regex>          -- one or more targetURLs
    ct_switch  <source_ct, dest_ct> -- one or more CTs
}

```

## EXAMPLE

```

ct_map {
    target_url '^coap://class-1-device/.*$'
    ct_switch  */xml    application/exi
}

```



Authors' Addresses

Angelo P. Castellani  
University of Padova  
Via Gradenigo 6/B  
Padova 35131  
Italy

Email: [angelo@castellani.net](mailto:angelo@castellani.net)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: [salvatore.loreto@ericsson.com](mailto:salvatore.loreto@ericsson.com)

Akbar Rahman  
InterDigital Communications, LLC  
1000 Sherbrooke Street West  
Montreal H3A 3G4  
Canada

Phone: +1 514 585 0761  
Email: [Akbar.Rahman@InterDigital.com](mailto:Akbar.Rahman@InterDigital.com)

Thomas Fossati  
KoanLogic  
Via di Sabbiuono 11/5  
Bologna 40136  
Italy

Phone: +39 051 644 82 68  
Email: [tho@koanlogic.com](mailto:tho@koanlogic.com)

Esko Dijk  
Philips Research

Email: [esko.dijk@philips.com](mailto:esko.dijk@philips.com)

CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: August 29, 2013

A. Castellani  
University of Padova  
S. Loreto  
Ericsson  
A. Rahman  
InterDigital Communications, LLC  
T. Fossati  
KoanLogic  
E. Dijk  
Philips Research  
February 25, 2013

Best Practices for HTTP-CoAP Mapping Implementation  
draft-castellani-core-http-mapping-07

Abstract

This draft provides reference information for HTTP-CoAP protocol translation proxy implementors, focusing primarily on the reverse proxy case. It details deployment options, discusses possible approaches for URI mapping, and provides a set of guidelines and considerations related to protocol translation.

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 August 29, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .                     | 3  |
| 2. Terminology . . . . .                      | 3  |
| 3. Cross-Protocol Usage of URIs . . . . .     | 4  |
| 4. HTTP to CoAP URI Mapping . . . . .         | 5  |
| 4.1. Embedded Mapping . . . . .               | 5  |
| 4.2. Homogeneous Mapping . . . . .            | 5  |
| 4.3. Scheme Security Mapping . . . . .        | 6  |
| 5. HTTP-CoAP Reverse Proxy . . . . .          | 6  |
| 5.1. Proxy Placement . . . . .                | 7  |
| 5.2. Response Code Translations . . . . .     | 8  |
| 5.3. Media Type Translations . . . . .        | 10 |
| 5.4. Caching and Congestion Control . . . . . | 11 |
| 5.5. Cache Refresh via Observe . . . . .      | 11 |
| 5.6. Use of CoAP Blockwise Transfer . . . . . | 12 |
| 5.7. Security Translation . . . . .           | 13 |
| 5.8. Other guidelines . . . . .               | 13 |
| 6. IANA Considerations . . . . .              | 14 |
| 7. Security Considerations . . . . .          | 14 |
| 7.1. Traffic overflow . . . . .               | 14 |
| 7.2. Handling Secured Exchanges . . . . .     | 15 |
| 8. Acknowledgements . . . . .                 | 15 |
| 9. References . . . . .                       | 16 |
| 9.1. Normative References . . . . .           | 16 |
| 9.2. Informative References . . . . .         | 17 |
| Authors' Addresses . . . . .                  | 17 |

## 1. Introduction

CoAP [I-D.ietf-core-coap] has been designed with the twofold aim to be an application protocol specialized for constrained environments and to be easily used in REST architectures such as the Web. The latter goal has led to define CoAP to easily interoperate with HTTP [RFC2616] through an intermediary proxy which performs cross-protocol conversion.

Section 10 of [I-D.ietf-core-coap] describes the fundamentals of the CoAP-HTTP (and vice-versa) cross-protocol mapping process. However, implementing such a cross-protocol proxy can be complex, and many details regarding its internal procedures and design choices require further elaboration. Therefore a first goal of this document is to provide more detailed information to proxy designers and implementers, to help implement proxies that correctly inter-work with other CoAP and HTTP client/server implementations that adhere to the HTTP and CoAP specifications.

The second goal of this informational document is to define a consistent set of guidelines that a HTTP-to-CoAP proxy implementation MAY adhere to. The main reason of adhering to such guidelines is to reduce variation between proxy implementations, thereby increasing interoperability. (As an example use case, a proxy conforming to these guidelines made by vendor A can be easily replaced by a proxy from vendor B that also conforms to the guidelines.)

This draft is organized as follows:

- o Section 2 describes terminology to identify proxy types, mapping approaches and proxy deployments;
- o Section 3 discusses how URIs refer to resources independent of access protocols;
- o Section 5 analyzes the mapping that allows HTTP clients to contact CoAP servers;
- o Section 7 discusses possible security impact related to HTTP/CoAP cross-protocol mapping.

## 2. Terminology

This document assumes readers are familiar with the terms Reverse Proxy as defined in [I-D.ietf-httpbis-pl-messaging] and Interception Proxy as defined in [RFC3040]. In addition, the following terms are defined:

Cross-Protocol Proxy (or Cross Proxy): is a proxy performing a cross-protocol mapping, in the context of this document a HTTP-CoAP (HC) mapping. A Cross-Protocol Proxy can behave as a Forward Proxy, Reverse Proxy or Interception Proxy. Note: In this document we focus on the Reverse Proxy mode of the Cross-Protocol Proxy.

Forward Proxy: a message forwarding agent that is selected by the client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and to attempt to satisfy those requests via translation to the protocol indicated by the absolute URI. The user decides (is willing to) use the proxy as the forwarding/dereferencing agent for a predefined subset of the URI space.

Reverse Proxy: a receiving agent that acts as a layer above some other server(s) and translates the received requests to the underlying server's protocol. It behaves as an origin (HTTP) server on its connection towards the (HTTP) client and as a (CoAP) client on its connection towards the (CoAP) origin server. The (HTTP) client uses the "origin-form" [I-D.ietf-httpbis-pl-messaging] as a request-target URI.

Reverse and Forward proxies are technically very similar, with main differences being that the former appears to a client as an origin server while the latter does not, and that clients may be unaware they are communicating with a proxy.

Placement terms: a server-side (SS) proxy is placed in the same network domain as the server; conversely a client-side (CS) proxy is in the same network domain as the client. In any other case than SS or CS, the proxy is said to be External (E).

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

### 3. Cross-Protocol Usage of URIs

A Uniform Resource Identifier (URI) provides a simple and extensible method for identifying a resource. It enables uniform identification of resources via a separately defined extensible set of naming schemes [RFC3986].

URIs are formed of at least three components: scheme, authority and path. The scheme often corresponds to the protocol used to access the resource. However, as noted in Section 1.2.2 of [RFC3986] the

scheme does not imply that a particular protocol is used to access the resource. So, we can define the same resource to be accessible by different protocols i.e. the resource can have cross-protocol URIs referring to it.

HTTP clients typically only support 'http' and 'https' schemes. Therefore, they cannot directly access CoAP servers (which support 'coap' and/or 'coaps'). In this situation, communication is enabled by a Cross-Protocol Proxy, as shown in Figure 1, supporting URI mapping features. Such features are discussed in the following section.

#### 4. HTTP to CoAP URI Mapping

Assume that a HTTP client wants to access a CoAP resource and indicates a target resource of "http://node.something.net/foobar" to a Forward cross proxy. A possible URI mapping done by the proxy could result in "coap://node.coap.something.net/foo".

As shown in the above example, in a cross-protocol URI the scheme, authority and path parts of the URI may all change. The process of providing cross-protocol URIs may be complex, since a mechanism to statically or dynamically (e.g., discovery) map the URI is needed.

Two simple static URI mapping solutions are proposed in the following subsections. Note that other mapping approaches are possible as well.

##### 4.1. Embedded Mapping

In an embedded mapping approach, the HTTP URI has embedded inside it the authority and path part of the CoAP URI.

Example: The CoAP resource "//node.coap.something.net/foo" can be accessed by an HTTP client by inserting in the request "http://hc-proxy.something.net/coap/node.coap.something.net/foo". The Cross-Protocol Proxy then maps the URI to "coap://node.coap.something.net/foo"

##### 4.2. Homogeneous Mapping

In a homogeneous mapping approach, only the scheme portion of the URI needs to be mapped. The rest of the URI (i.e. authority, path, etc.) remains unchanged.

Example: The CoAP resource "coap://node.coap.something.net/foo" can be accessed by an HTTP client by requesting

"http://node.coap.something.net/foo". The Cross-Protocol Proxy receiving the request is responsible to map the URI to "coap://node.coap.something.net/foo"

Background info: The assumption in this case is that the HTTP client would be able to successfully resolve "node.coap.something.net" using DNS infrastructure to return the IP address of the HC proxy. Most likely this would be through a two step DNS lookup where the first DNS lookup would resolve "something.net" using public DNS infrastructure. Then the second DNS lookup on the subdomain "coap" and the host "node" would typically be resolved by a DNS server operated by the owner of domain "something.net". So this domain owner can manage its own internal node names and subdomain allocation which would correspond to the CoAP namespace

#### 4.3. Scheme Security Mapping

In general, regardless of the URI mapping scheme used in the Cross-Protocol Proxy, an "https" request SHOULD be translated to a "coaps" request. The exception case being cases where security on the CoAP side is not needed because the network is well enough protected already by other means (e.g. strong link-layer security, or the CoAP network runs inside a firewalled network, etc.).

#### 5. HTTP-CoAP Reverse Proxy

A HTTP-CoAP Reverse Cross-Protocol Proxy is accessed by web clients only supporting HTTP, and handles their requests by mapping these to CoAP requests, which are forwarded to CoAP servers; and mapping back the received CoAP responses to HTTP. This mechanism is transparent to the client, which may assume that it is communicating with the intended target HTTP server. In other words, the client accesses the proxy as an origin server using the "origin-form" [I-D.ietf-httpbis-pl-messaging] as a Request Target.

Normative requirements on the translation of HTTP requests to CoAP and of the CoAP responses back to HTTP responses are defined in Section 10.2 of [I-D.ietf-core-coap]. However, that section only considers the case of a HTTP-CoAP Forward Cross-Protocol Proxy in which a client explicitly indicates it targets a request to a CoAP server, and does not cover all aspects of proxy implementation in detail. The present section provides guidelines and more details for the implementation of a Reverse Cross-Protocol Proxy, which MAY be followed in addition to the normative requirements.

Translation of unicast HTTP requests into multicast CoAP requests is currently out of scope since in a reverse proxy scenario a HTTP

client typically expects to receive a single response, not multiple. However a Cross-Protocol Proxy MAY include custom application-specific functions to generate a multicast CoAP request based on a unicast HTTP request and aggregate multiple CoAP responses into a single HTTP response.

Note that the guidelines in this section also apply to an HTTP-CoAP Intercepting Cross-Protocol Proxy.

### 5.1. Proxy Placement

Typically, a Cross-Protocol Proxy is located at the edge of the constrained network. See Figure 1. The arguments supporting server-side (SS) placement are the following:

Caching: Efficient caching requires that all request traffic to a CoAP server is handled by the same proxy which receives HTTP requests from multiple source locations. This maximally reduces the load on (constrained) CoAP servers.

Multicast: To support CoAPs use of local-multicast functionalities available in a constrained network, the Cross-Protocol Proxy requires a network interface directly attached to the constrained network.

TCP/UDP: Translation between HTTP and CoAP requires also TCP/UDP translation; TCP may be the preferred way for communicating with the constrained network due to its reliability or due to intermediate gateways configured to block UDP traffic.

Arguments against SS placement, in favor of client-side (CS), are:

Scalability: A solution where a single SS proxy has to manage numerous open TCP/IP connections to a large number of HTTP clients is not scalable. (Unless multiple SS proxies are employed with a load-balancing mechanism, which adds complexity.)



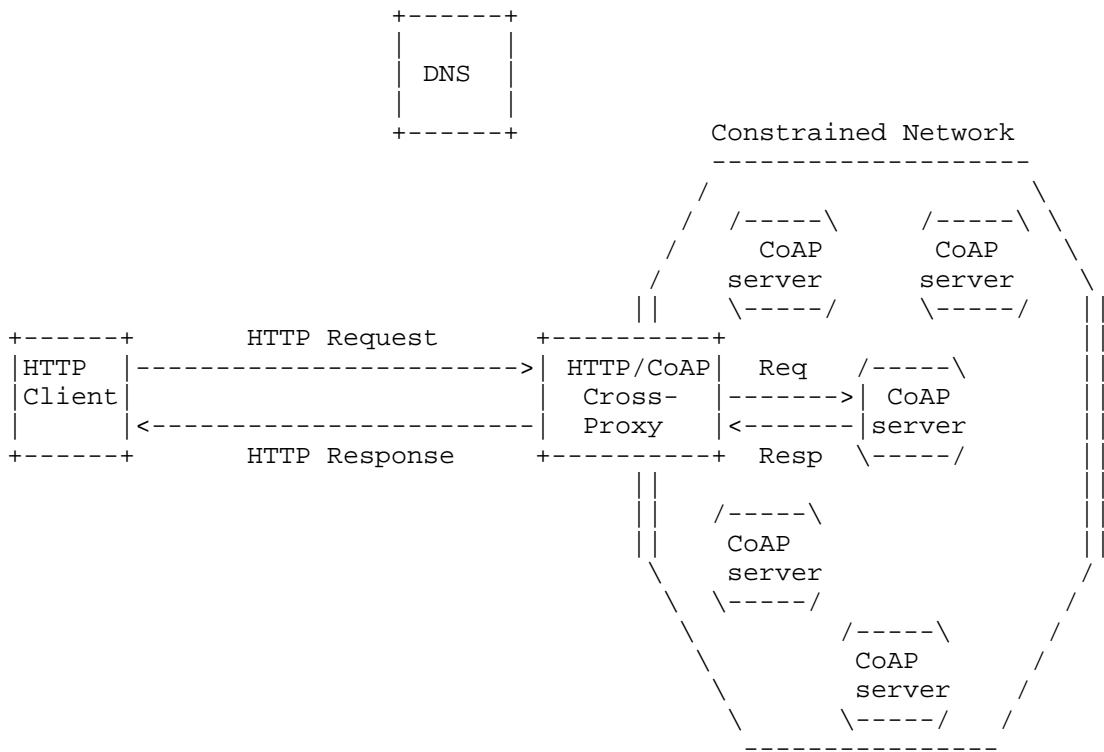


Figure 1: Reverse Cross-Protocol Proxy Deployment Scenario

5.2. Response Code Translations

Table 1 defines all possible CoAP responses along with the HTTP response to which each CoAP response SHOULD be translated. This table complies with the Section 10.2 requirements of [I-D.ietf-core-coap] and is intended to cover all possible cases. Multiple appearances of a HTTP status code in the second column indicates multiple equivalent HTTP responses are possible, depending on the conditions cited in the Notes (third column).

| CoAP Response Code            | HTTP Status Code            | Notes |
|-------------------------------|-----------------------------|-------|
| 2.01 Created                  | 201 Created                 | 1     |
| 2.02 Deleted                  | 200 OK                      | 2     |
|                               | 204 No Content              | 2     |
| 2.03 Valid                    | 304 Not Modified            | 3     |
|                               | 200 OK                      | 4     |
| 2.04 Changed                  | 200 OK                      | 2     |
|                               | 204 No Content              | 2     |
| 2.05 Content                  | 200 OK                      |       |
| 4.00 Bad Request              | 400 Bad Request             |       |
| 4.01 Unauthorized             | 400 Bad Request             | 5     |
| 4.02 Bad Option               | 400 Bad Request             | 6     |
| 4.03 Forbidden                | 403 Forbidden               |       |
| 4.04 Not Found                | 404 Not Found               |       |
| 4.05 Method Not Allowed       | 400 Bad Request             | 7     |
| 4.06 Not Acceptable           | 406 Not Acceptable          |       |
| 4.12 Precondition Failed      | 412 Precondition Failed     |       |
| 4.13 Request Entity Too Large | 413 Request Repr. Too Large |       |
| 4.15 Unsupported Media Type   | 415 Unsupported Media Type  |       |
| 5.00 Internal Server Error    | 500 Internal Server Error   |       |
| 5.01 Not Implemented          | 501 Not Implemented         |       |
| 5.02 Bad Gateway              | 502 Bad Gateway             |       |
| 5.03 Service Unavailable      | 503 Service Unavailable     | 8     |
| 5.04 Gateway Timeout          | 504 Gateway Timeout         |       |
| 5.05 Proxying Not Supported   | 502 Bad Gateway             | 9     |

Table 1: HTTP-CoAP Response Mapping

## Notes:

1. A CoAP server may return an arbitrary format payload along with this response. This payload SHOULD be returned as entity in the HTTP 201 response. Section 7.3.2 of [I-D.ietf-httpbis-p2-semantics] does not put any requirement on the format of the payload. (In the past, [RFC2616] did.)
2. The HTTP code is 200 or 204 respectively for the case that a CoAP server returns a payload or not. [I-D.ietf-httpbis-p2-semantics] Section 5.3 requires code 200 in case a representation of the action result is returned for DELETE, POST and PUT and code 204 if not. Hence, a proxy SHOULD transfer any CoAP payload contained in a 2.02 response to the HTTP client in a 200 OK response.

3. A CoAP 2.03 (Valid) response only (1) confirms that the request ETag is valid and (2) provides a new Max-Age value. HTTP 304 (Not Modified) also updates some header fields of a stored response. A non-caching proxy may not have enough information to fill in the required values in the HTTP 304 (Not Modified) response, so it may not be advisable for a non-caching proxy to provoke the 2.03 (Valid) response by forwarding an ETag. A caching proxy will fill the information out of the cache.
4. A 200 response to a CoAP 2.03 occurs only when the proxy is caching and translated a HTTP request (without validation request) to a CoAP request that includes validation, for efficiency. The proxy receiving 2.03 updates the freshness of the cached representation and returns the entire representation to the HTTP client.
5. The HTTP code 401 Unauthorized MUST NOT be used, as long as in CoAP there is no equivalent defined of the required WWW-Authenticate header (Section 3.1 of [I-D.ietf-httpbis-p7-auth]).
6. In some cases a proxy receiving 4.02 may retry the request with less CoAP Options in the hope that the server will understand the newly formulated request. For example, if the proxy tried using a Block Option which was not recognized by the CoAP server it may retry without that Block Option.
7. The HTTP code "405 Method Not Allowed" MUST NOT be used since CoAP does not provide enough information to determine a value for the required "Allow" response-header field.
8. The value of the HTTP "Retry-After" response-header field is taken from the value of the CoAP Max-Age Option, if present.
9. This CoAP response can only happen if the proxy itself is configured to use a CoAP Forward Proxy to execute some, or all, of its CoAP requests.

### 5.3. Media Type Translations

A Cross-Protocol Proxy translates a media type string, carried in a HTTP Content-Type header in a request, to a CoAP Content-Format Option with the equivalent numeric value. The media types supported by CoAP are defined in the CoAP Content-Format Registry. Any HTTP request with a Content-Type for which the proxy does not know an equivalent CoAP Content-Format number, MUST lead to HTTP response 415 (Unsupported Media Type).

Also, a CoAP Content-Format value in a response is translated back to

the equivalent HTTP Content-Type. If a proxy receives a CoAP Content-Format value that it does not recognize (e.g. because the value is IANA-registered after the proxy software was deployed), and is unable to look up the equivalent HTTP Content-Type on the fly, the proxy SHOULD return an HTTP entity (payload) without Content-Type header (complying to Section 3.1.1.5 of [I-D.ietf-httpbis-p2-semantics]).

#### 5.4. Caching and Congestion Control

A Cross-Protocol Proxy SHOULD limit the number of requests to CoAP servers by responding, where applicable, with a cached representation of the resource.

Duplicate idempotent pending requests by a Cross-Protocol Proxy to the same CoAP resource SHOULD in general be avoided, by duplexing the response to the requesting HTTP clients without duplicating the CoAP request.

If the HTTP client times out and drops the HTTP session to the Cross-Protocol Proxy (closing the TCP connection) after the HTTP request was made, a Cross-Protocol Proxy SHOULD wait for the associated CoAP response and cache it if possible. Further requests to the Cross-Protocol Proxy for the same resource can use the result present in cache, or, if a response has still to come, the HTTP requests will wait on the open CoAP session.

According to [I-D.ietf-core-coap], a proxy MUST limit the number of outstanding interactions to a given CoAP server to NSTART. To limit the amount of aggregate traffic to a constrained network, the Cross-Protocol Proxy SHOULD also pose a limit to the number of concurrent CoAP requests pending on the same constrained network; further incoming requests MAY either be queued or dropped (returning 503 Service Unavailable). This limit and the proxy queueing/dropping behavior SHOULD be configurable. In order to efficiently apply this congestion control, the Cross-Protocol Proxy SHOULD be placed.

Resources experiencing a high access rate coupled with high volatility MAY be observed [I-D.ietf-core-observe] by the Cross-Protocol Proxy to keep their cached representation fresh while minimizing the number CoAP messages. See Section 5.5.

#### 5.5. Cache Refresh via Observe

There are cases where using the CoAP observe protocol [I-D.ietf-core-observe] to handle proxy cache refresh is preferable to the validation mechanism based on ETag as defined in [I-D.ietf-core-coap]. Such scenarios include, but are not limited

to, sleepy nodes -- with possibly high variance in requests' distribution -- which would greatly benefit from a server driven cache update mechanism. Ideal candidates would also be crowded or very low throughput networks, where reduction of the total number of exchanged messages is an important requirement.

This subsection aims at providing a practical evaluation method to decide whether the refresh of a cached resource R is more efficiently handled via ETag validation or by establishing an observation on R.

Let  $T_R$  be the mean time between two client requests to resource R, let  $F_R$  be the freshness lifetime of R representation, and let  $M_R$  be the total number of messages exchanged towards resource R. If we assume that the initial cost for establishing the observation is negligible, an observation on R reduces  $M_R$  iff  $T_R < 2 * F_R$  with respect to using ETag validation, that is iff the mean arrival time of requests for resource R is greater than half the refresh rate of R.

When using observations  $M_R$  is always upper bounded by  $2 * F_R$ : in the constrained network no more than  $2 * F_R$  messages will be generated towards resource R.

#### 5.6. Use of CoAP Blockwise Transfer

A Cross-Protocol Proxy SHOULD support CoAP blockwise transfers [I-D.ietf-core-block] to allow transport of large CoAP payloads while avoiding excessive link-layer fragmentation in LLNs, and to cope with small datagram buffers in CoAP end-points as described in [I-D.ietf-core-coap] Section 4.6.

A Cross-Protocol Proxy SHOULD attempt to retry a payload-carrying CoAP PUT or POST request with blockwise transfer if the destination CoAP server responded with 4.13 (Request Entity Too Large) to the original request. A Cross-Protocol Proxy SHOULD attempt to use blockwise transfer when sending a CoAP PUT or POST request message that is larger than a value `BLOCKWISE_THRESHOLD`. The value of `BLOCKWISE_THRESHOLD` MAY be implementation-specific, for example calculated based on a known or typical UDP datagram buffer size for CoAP end-points, or set to N times the size of a link-layer frame where e.g.  $N=5$ , or preset to a known IP MTU value, or set to a known Path MTU value. The value `BLOCKWISE_THRESHOLD` or parameters from which it is calculated SHOULD be configurable in a proxy implementation.

The Cross-Protocol Proxy SHOULD detect CoAP end-points not supporting blockwise transfers by checking for a 4.02 (Bad Option) response returned by an end-point in response to a CoAP request with a Block\*

Option. This allows the Cross-Protocol Proxy to be more efficient, not attempting repeated blockwise transfers to CoAP servers that do not support it. However if a request payload is too large to be sent as a single CoAP request and blockwise transfer would be unavoidable, the proxy still SHOULD attempt blockwise transfer on such an endpoint before returning 413 (Request Entity Too Large) to the HTTP client.

For improved latency a cross proxy MAY initiate a blockwise CoAP request triggered by an incoming HTTP request even when the HTTP request message has not yet been fully received, but enough data has been received to send one or more data blocks to a CoAP server already. This is particularly useful on slow client-to-proxy connections.

#### 5.7. Security Translation

A HC proxy SHOULD implement explicit rules for security context translations. A translation may involve e.g. applying a rule that any "https" request is translated to a "coaps" request, or e.g. applying a rule that a "https" request is translated to an unsecured "coap" request. Another rule could specify the security policy and parameters used for DTLS connections. Such rules will largely depend on the application and network context in which a proxy is applied. To enable widest possible use of a proxy implementation, these rules SHOULD be configurable in a HC proxy.

#### 5.8. Other guidelines

For long delays of a CoAP server, the HTTP client or any other proxy in between MAY timeout. Further discussion of timeouts in HTTP is available in Section 6.2.4 of [I-D.ietf-httpbis-pl-messaging].

A cross proxy MUST define an internal timeout for each pending CoAP request, because the CoAP server may silently die before completing the request. The timeout value SHOULD be approximately less than or equal to MAX\_RTT defined in [I-D.ietf-core-coap].

When the DNS protocol is not used between CoAP nodes in a constrained network, defining valid FQDN (i.e., DNS entries) for constrained CoAP servers, where possible, MAY help HTTP clients to access the resources offered by these servers via a HC proxy.

HTTP connection pipelining (section 6.2.2.1 of [I-D.ietf-httpbis-pl-messaging]) MAY be supported by the proxy and is transparent to the CoAP network: the HC cross proxy will sequentially serve the pipelined requests by issuing different CoAP requests.

## 6. IANA Considerations

This memo includes no request to IANA.

## 7. Security Considerations

The security concerns raised in Section 15.7 of [RFC2616] also apply to the cross proxy scenario. In fact, the cross proxy is a trusted (not rarely a transparently trusted) component in the network path.

The trustworthiness assumption on the cross proxy cannot be dropped. Even if we had a blind, bi-directional, end-to-end, tunneling facility like the one provided by the CONNECT method in HTTP, and also assuming the existence of a DTLS-TLS transparent mapping, the two tunneled ends should be speaking the same application protocol, which is not the case. Basically, the protocol translation function is a core duty of the cross proxy that can't be removed, and makes it a necessarily trusted, impossible to bypass, component in the communication path.

A reverse proxy deployed at the boundary of a constrained network is an easy single point of failure for reducing availability. As such, a special care should be taken in designing, developing and operating it, keeping in mind that, in most cases, it could have fewer limitations than the constrained devices it is serving.

The following sub paragraphs categorize and argue about a set of specific security issues related to the translation, caching and forwarding functionality exposed by a cross proxy module.

### 7.1. Traffic overflow

Due to the typically constrained nature of CoAP nodes, particular attention SHOULD be posed in the implementation of traffic reduction mechanisms (see Section 5.4), because inefficient implementations can be targeted by unconstrained Internet attackers. Bandwidth or complexity involved in such attacks is very low.

An amplification attack to the constrained network may be triggered by a multicast request generated by a single HTTP request mapped to a CoAP multicast resource, as considered in Section TBD of [I-D.ietf-core-coap].

The impact of this amplification technique is higher than an amplification attack carried out by a malicious constrained device (e.g. ICMPv6 flooding, like Packet Too Big, or Parameter Problem on a multicast destination [RFC4732]), since it does not require direct

access to the constrained network.

The feasibility of this attack, disruptive in terms of CoAP server availability, can be limited by access controlling the exposed HTTP multicast resource, so that only known/authorized users access such URIs.

## 7.2. Handling Secured Exchanges

It is possible that the request from the client to the cross proxy is sent over a secured connection. However, there may or may not exist a secure connection mapping to the other protocol. For example, a secure distribution method for multicast traffic is complex and MAY not be implemented (see [I-D.ietf-core-groupcomm]).

By default, a cross proxy SHOULD reject any secured client request if there is no configured security policy mapping. This recommendation MAY be relaxed in case the destination network is believed to be secured by other, complementary, means. E.g.: assumed that CoAP nodes are isolated behind a firewall (e.g. as the SS cross proxy deployment shown in Figure 1), the cross proxy may be configured to translate the incoming HTTPS request using plain CoAP (i.e. NoSec mode.)

The HC URI mapping MUST NOT map to HTTP (see Section 4) a CoAP resource intended to be accessed only using HTTPS.

A secured connection that is terminated at the cross proxy, i.e. the proxy decrypts secured data locally, raises an ambiguity about the cacheability of the requested resource. The cross proxy SHOULD NOT cache any secured content to avoid any leak of secured information. However in some specific scenario, a security/efficiency trade-off could motivate caching secured information; in that case the caching behavior MAY be tuned to some extent on a per-resource basis.

## 8. Acknowledgements

An initial version of the table found in Section 5.2 has been provided in revision -05 of [I-D.ietf-core-coap]. Special thanks to Peter van der Stok for countless comments and discussions on this document, that contributed to its current structure and text.

Thanks to Carsten Bormann, Zach Shelby, Michele Rossi, Nicola Bui, Michele Zorzi, Klaus Hartke, Cullen Jennings, Kepeng Li, Brian Frank, Peter Saint-Andre, Kerry Lynn, Linyi Tian, Dorothy Gellert, Francesco Corazza for helpful comments and discussions that have shaped the document.



The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement n. [251557].

## 9. References

### 9.1. Normative References

- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",  
draft-ietf-core-block-10 (work in progress), October 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.ietf-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP",  
draft-ietf-core-groupcomm-05 (work in progress),  
February 2013.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-07 (work in progress),  
October 2012.
- [I-D.ietf-httpbis-p1-messaging]  
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol  
(HTTP/1.1): Message Syntax and Routing",  
draft-ietf-httpbis-p1-messaging-22 (work in progress),  
February 2013.
- [I-D.ietf-httpbis-p2-semantics]  
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol  
(HTTP/1.1): Semantics and Content",  
draft-ietf-httpbis-p2-semantics-22 (work in progress),  
February 2013.
- [I-D.ietf-httpbis-p7-auth]  
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol  
(HTTP/1.1): Authentication", draft-ietf-httpbis-p7-auth-22  
(work in progress), February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.

- [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, June 1999.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

## 9.2. Informative References

- [I-D.bormann-core-simple-server-discovery] Bormann, C., "CoRE Simple Server Discovery", draft-bormann-core-simple-server-discovery-01 (work in progress), March 2012.
- [I-D.shelby-core-resource-directory] Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-04 (work in progress), July 2012.
- [RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.
- [RFC4732] Handley, M., Rescorla, E., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, December 2006.

## Authors' Addresses

Angelo P. Castellani  
University of Padova  
Via Gradenigo 6/B  
Padova 35131  
Italy  
  
Email: [angelo@castellani.net](mailto:angelo@castellani.net)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland  
  
Email: [salvatore.loreto@ericsson.com](mailto:salvatore.loreto@ericsson.com)

Akbar Rahman  
InterDigital Communications, LLC  
1000 Sherbrooke Street West  
Montreal H3A 3G4  
Canada

Phone: +1 514 585 0761  
Email: Akbar.Rahman@InterDigital.com

Thomas Fossati  
KoanLogic  
Via di Sabbiano 11/5  
Bologna 40136  
Italy

Phone: +39 051 644 82 68  
Email: tho@koanlogic.com

Esko Dijk  
Philips Research  
High Tech Campus 34  
Eindhoven 5656 AE  
The Netherlands

Email: esko.dijk@philips.com



CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: December 11, 2014

E. Dijk, Ed.  
Philips Research  
A. Rahman, Ed.  
InterDigital Communications, LLC  
June 9, 2014

Miscellaneous CoAP Group Communication Topics  
draft-dijk-core-groupcomm-misc-06

Abstract

This document contains miscellaneous text around the topic of group communication for the Constrained Application Protocol (CoAP). The intent of this document is to keep track of useful ideas while the main CoRE WG Group Communication draft goes through the RFC publication process. These ideas may be then be used as input to future standardization in the CoRE or other related WGs.

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 December 11, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .   | 3  |
| 2. Potential Solutions for Group Communication . . . . .              | 3  |
| 3. Use Cases . . . . .  | 3  |
| 4. Requirements . . . . .   | 4  |
| 4.1. Background . . . . .   | 4  |
| 4.2. General Requirements . . . . .                                   | 5  |
| 4.3. Security Requirements . . . . .                                  | 6  |
| 5. Group Communication Solutions . . . . .                            | 8  |
| 5.1. IP Multicast Transmission Methods . . . . .                      | 8  |
| 5.1.1. Serial unicast . . . . .                                       | 8  |
| 5.1.2. Unreliable IP Multicast . . . . .                              | 8  |
| 5.1.3. Reliable IP Multicast . . . . .                                | 8  |
| 5.2. Overlay Multicast . . . . .                                      | 9  |
| 5.3. CoAP Application Layer Group Management . . . . .                | 10 |
| 6. DNS-SD Based Group Resource Manipulation . . . . .                 | 13 |
| 7. Group Discovery and Member Discovery . . . . .                     | 13 |
| 7.1. DNS-SD . . . . .   | 13 |
| 7.2. CoRE Resource Directory . . . . .                                | 14 |
| 8. Deployment Guidelines . . . . .                                    | 14 |
| 8.1. Overview . . . . .   | 14 |
| 8.2. Implementation in Target Network Topologies . . . . .            | 15 |
| 8.2.1. Single LLN Topology . . . . .                                  | 15 |
| 8.2.2. Single LLN with Backbone Topology . . . . .                    | 17 |
| 8.2.3. Multiple LLNs with Backbone Topology . . . . .                 | 19 |
| 8.2.4. LLN(s) with Multiple 6LBRs . . . . .                           | 19 |
| 8.2.5. Conclusions . . . . .  | 19 |
| 8.3. Implementation Considerations . . . . .                          | 20 |
| 8.3.1. MLD Implementation on LLNs and MLD alternatives . . . . .      | 20 |
| 8.3.2. 6LBR Implementation . . . . .                                  | 21 |
| 8.3.3. Backbone IP Multicast Infrastructure . . . . .                 | 21 |
| 9. Miscellaneous Topics . . . . .                                     | 22 |
| 9.1. CoAP Multicast and HTTP Unicast Interworking . . . . .           | 22 |
| 10. Acknowledgements . . . . .  | 24 |
| 11. IANA Considerations . . . . .                                     | 24 |
| 12. Security Considerations . . . . .                                 | 24 |
| 13. References . . . . .  | 25 |
| 13.1. Normative References . . . . .                                  | 25 |
| 13.2. Informative References . . . . .                                | 26 |
| Appendix A. Multicast Listener Discovery (MLD) . . . . .              | 28 |
| Appendix B. CoAP-Observe Alternative to Group Communication . . . . . | 29 |
| Authors' Addresses . . . . .  | 29 |

## 1. Introduction

This document contains miscellaneous text around the topic of group communication for the Constrained Application Protocol, CoAP [I-D.ietf-core-coap]. The first part of the document contains, for reference, text that was removed from the Group Communication for CoAP [I-D.ietf-core-groupcomm] draft and its predecessor [I-D.rahman-core-groupcomm]. The second part of the document contains text and/or functionality that may be considered for input to future standardization in the CoRE or other related WGs.

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 [RFC2119].

## 2. Potential Solutions for Group Communication

The classic concept of group communications is that of a single source distributing content to multiple destination recipients that are all part of a group. Before content can be distributed, there is a separate process to form the group. The source may be either a member or non-member of the group.

Group communication solutions have evolved from "bottom" to "top", i.e., from layer 2 (Media Access Control broadcast/multicast) and layer 3 (IP multicast) to application layer group communication, also referred to as application layer multicast. A study published in 2005 [Lao05] identified new solutions in the "middle" (referred to as overlay multicast) that utilize an infrastructure based on proxies.

Each of these classes of solutions may be compared [Lao05] using metrics such as link stress and level of host complexity [Banerjee01]. The results show for a realistic internet topology that IP Multicast is the most resource-efficient, with the downside being that it requires the most effort to deploy in the infrastructure. IP Multicast is the solution adopted by this draft for CoAP group communication.

## 3. Use Cases

CoAP group communication can be applied in the context of the following use cases:

- o Discovery of Resource Directory: discovering the local CoRE RD which contains links (URIs) to resources stored on other servers [RFC6690].

- o Lighting Control: synchronous operation of a group of IPv6-connected lights (e.g., 6LoWPAN [RFC4944] lights).
- o Parameter Update: updating parameters/settings simultaneously in a large group of devices in a building/campus control ([I-D.vanderstok-core-bc]) application.
- o Firmware Update: efficiently updating firmware simultaneously in a large group of devices in a building/campus control ([I-D.vanderstok-core-bc]) application. Here, the use of CoAP group communication could be realized via a multicast extension of CoAP blockwise transfer [I-D.ietf-core-block]. This use case and use of multicast is especially valuable if there are time constraints related to the software update for large groups of devices.
- o Group Status Report: requesting status information or event reports from a group of devices in a building/campus control application. In this use case, conditional reporting is required: only device that have events to report (as indicated by the request query) respond, others remain silent. This use case requires reliable CoAP group communication, which is currently not in CoRE WG scope.

#### 4. Requirements

Requirements that a CoAP group communication solution should fulfill can be found in existing documents ([RFC5867], [I-D.ietf-6lowpan-routing-requirements], [I-D.vanderstok-core-bc], and [I-D.shelby-core-coap-req]). Below, a set of high-level requirements is listed that a group communication solution should ideally fulfill. In practice, all these requirements can never be satisfied at once in an LLN context. Furthermore, different use cases will have different needs i.e. an elaboration of a subset of below requirements.

##### 4.1. Background

The requirements for CoAP are documented in [I-D.shelby-core-coap-req]. In this draft, we focus and expand discussions on the requirements pertaining to CoAP "group communication" and "multicast" support as stated in [I-D.shelby-core-coap-req]:

REQ 9: CoAP will support a non-reliable IP multicast message to be sent to a group of Devices to manipulate a resource on all the Devices simultaneously. The use of multicast to query and



advertise descriptions must be supported, along with the support of unicast responses.

Currently, the CoAP protocol [I-D.ietf-core-coap] supports unreliable IP multicast using UDP. It defines the unreliable multicast operation as follows in Section 4.5:

"CoAP supports sending messages to multicast destination addresses. Such multicast messages MUST be Non-Confirmable. Some mechanisms for avoiding congestion from multicast requests are being considered in [I-D.eggert-core-congestion-control]."

Additional requirements were introduced in [I-D.vanderstok-core-bc] driven by quality of experience issues in commercial lighting; the need for large numbers of devices to respond with near simultaneity to a command (multicast PUT), and for that command to be received reliably (reliable multicast).

#### 4.2. General Requirements

A CoAP group communication solution should (ideally) meet the following general requirements:

- GEN-REQ 1:      Optional Reliability: the application can select between unreliable group communication and reliable group communication.
- GEN-REQ 2:      Efficiency: delivers messages more efficiently than a "serial unicast" solution. Provides a balance between group data traffic and control overhead.
- GEN-REQ 3:      Low latency: deliver a message as quickly as possible.
- GEN-REQ 4:      Synchrony: allows near-simultaneous modification of a resource on all devices in a target group, providing a perceived effect of synchrony or simultaneity. For example a specified time span  $D$  such that a message is delivered to all destinations in a time interval  $[t, t+D]$ .
- GEN-REQ 5:      Ordering: message ordering may be required for reliable group communication use cases.
- GEN-REQ 6:      Security: see Section 4.3 for security requirements for group communication.
- GEN-REQ 7:      Flexibility: support for one or many source(s), both dense and sparse networks, for high or low listener

density, small or large number of groups, and multi-group membership.

- GEN-REQ 8: Robust group management: functionality to join groups, leave groups, view group membership, and persistent group membership in failure or sleeping node situations.
- GEN-REQ 9: Network layer independence: a solution is independent from specific unicast and/or IP multicast routing protocols.
- GEN-REQ 10: Minimal specification overhead: a group communication solution should preferably re-use existing/established (IETF) protocols that are suitable for LLN deployments, instead of defining new protocols from scratch.
- GEN-REQ 11: Minimal implementation overhead: e.g. a solution allows to re-use existing (software) components that are already present on constrained nodes such as (typical) 6LoWPAN/CoAP nodes.
- GEN-REQ 12: Mixed backbone/LLN topology support: a solution should work within a single LLN, and in combined LLN/backbone network topologies, including multi-LLN topologies. Both the senders and receivers of CoAP group messages may be attached to different network links or be part of different LLNs, possibly with routers or switches in between group members. In addition, different routing protocols may operate on the LLN and backbone networks. Preferably a solution also works with existing, common backbone IP infrastructure (e.g. switches or routers).
- GEN-REQ 13: CoAP Proxying support: a CoAP proxy can handle distribution of a message to a group on behalf of a (constrained) CoAP client.
- GEN-REQ 14: Suitable for operation on LLNs with constrained nodes.

#### 4.3. Security Requirements

Security for group communications at the IP level has been studied extensively in the IETF MSEC (Multicast Security) WG, and to a lesser extent in the IRTF SAMRG (Scalable Adaptive Multicast Research Group). In particular, [RFC3740], [RFC5374] and [RFC4046] are very instructive. A set of requirements for securing group communications in CoAP were derived from a study of these previous investigations as

well as understanding of CoAP specific needs. These are listed below.

A CoAP group communication solution should (ideally) meet the following security requirements:

- SEC-REQ 1:    Group communications data encryption: Important CoAP group communications shall be encrypted (using a group key) to preserve confidentiality. It shall also be possible to send CoAP group communications in the clear (i.e. unencrypted) for low value data.
- SEC-REQ 2:    Group communications source data authentication: Important CoAP group communications shall be authenticated by verifying the source of the data (i.e. that it was generated by a given and trusted group member). It shall also be possible to send unauthenticated CoAP group communications for low value data.
- SEC-REQ 3:    Group communications limited data authentication: Less important CoAP group communications shall be authenticated by simply verifying that it originated from one of the group members (i.e. without explicitly identifying the source node). This is a weaker requirement (but simpler to implement) than REQ2. It shall also be possible to send unauthenticated CoAP group communications for low value data.
- SEC-REQ 4:    Group key management: There shall be a secure mechanism to manage the cryptographic keys (e.g. generation and distribution) belonging to the group; the state (e.g. current membership) associated with the keys; and other security parameters.
- SEC-REQ 5:    Use of Multicast IPSec: The CoAP protocol [I-D.ietf-core-coap] allows IPSec to be used as one option to secure CoAP. If IPSec is used as a way to security CoAP communications, then multicast IPSec [RFC5374] should be used for securing CoAP group communications.
- SEC-REQ 6:    Independence from underlying routing security: CoAP group communication security shall not be tied to the security of underlying routing and distribution protocols such as PIM [RFC4601] and RPL [RFC6550]. Insecure or inappropriate routing (including IP multicast routing) may cause loss of data to CoAP but

will not affect the authenticity or secrecy of CoAP group communications.

SEC-REQ 7:      Interaction with HTTPS: The security scheme for CoAP group communications shall account for the fact that it may need to interact with HTTPS (Hypertext Transfer Protocol Secure) when a transaction involves a node in the general Internet (non-constrained network) communicating via a HTTP-CoAP proxy.

## 5. Group Communication Solutions

This section includes the text that describes the solutions of IP multicast, overlay multicast, and application layer group communication which were removed from [I-D.rahman-core-groupcomm] version 07 when the text was transferred to [I-D.ietf-core-groupcomm].

### 5.1. IP Multicast Transmission Methods

#### 5.1.1. Serial unicast

Even in systems that generally support IP Multicast, there may be certain data links (or transports) that don't support IP multicast. For those links a serial unicast alternative must be provided. This implies that it should be possible to enumerate the members of a group, in order to determine the correct unicast destinations.

#### 5.1.2. Unreliable IP Multicast

The CoRE WG charter specified support for non-reliable IP multicast. In the current CoAP protocol design [I-D.ietf-core-coap], unreliable multicast is realized by the source sending Non-Confirmable messages to a multicast IP address. IP Multicast (using UDP) in itself is unreliable, unless specific reliability features are added to it.

#### 5.1.3. Reliable IP Multicast

[TBD: This is a difficult problem. Need to investigate the benefits of repeating MGET and MPUT requests (saturation) to get "Pretty Good Reliability". Use the same MID or a new MID for repeated requests? Carsten suggests the use of bloom filters to suppress duplicate responses.

One could argue that non-idempotent operations (POST) cannot be supported without a *truly* reliable multicast protocol. However, is this the case? If a multicast POST request is sent repeatedly with the same Message ID (MID), then CoAP nodes that already received it

once will ignore duplicates. Sending with Message ID is supported in CoAP for Non-Confirmable messages (thus including multicast messages) as per [I-D.ietf-core-coap] section 4.2. ]

Reliable multicast supports guaranteed delivery of messages to a group of nodes. The following specifies the requirements as was proposed originally in version 01 of [I-D.vanderstok-core-bc]:

- o Validity - If sender sends a message, *m*, to a group, *g*, of destinations, a path exists between sender and destinations, and the sender and destinations are correct, all destinations in *g* eventually receive *m*.
- o Integrity - destination receives *m* at most once from sender and only if sender sent *m* to a group including destination.
- o Agreement - If a correct destination of *g* receives *m*, then all correct destinations of *g* receive *m*.
- o Timeliness - For real-time control of devices, there is a known constant *D* such that if *m* is sent at time *t*, no correct destination receives *m* after *t*+*D*.

There are various approaches to achieve reliability, such as

- o Destination node sends response: a destination sends a CoAP Response upon multicast Request reception (it SHOULD be a Non-Confirmable response). The source node may retry a request to destination nodes that did not respond in time with a CoAP response.
- o Route redundancy
- o Source node transmits multiple times (destinations do not respond)

## 5.2. Overlay Multicast

An alternative group communication solution (to IP Multicast) is an "overlay multicast" approach. We define an overlay multicast as one that utilizes an infrastructure based on proxies (rather than an IP router based IP multicast backbone) to deliver IP multicast packets to end devices. MLD ([RFC3810]) has been selected as the basis for multicast support by the ROLL working group for the RPL routing protocol. Therefore, it is proposed that "IGMP/MLD Proxying" [RFC4605] be used as a basis for an overlay multicast solution for CoAP.

Specifically, a CoAP proxy [I-D.ietf-core-coap] may also contain an MLD Proxy function. All CoAP devices that want to join a given IP multicast group would then send an MLD Join to the CoAP (MLD) proxy. Thereafter, the CoAP (MLD) proxy would be responsible for delivering any IP multicast message to the subscribed CoAP devices. This will require modifications to the existing [RFC4605] functionality.

Note that the CoAP (MLD) proxy may or may not be connected to an external IP multicast enabled backbone. The key function for the CoAP (MLD) proxy is to distribute CoAP generated multicast packets even in the absence of router support for multicast.

### 5.3. CoAP Application Layer Group Management

Another alternative solution (to IP Multicast and Overlay Multicast) is to define CoAP application level group management primitives. Thus, CoAP can support group management features without need for any underlying IP multicast support.

Interestingly, such group management primitives could also be offered even if there is underlying IP multicast support. This is useful because IP multicast inherently does not support the concept of a group with managed members, while a managed group may be required for some applications.

The following group management primitives are in general useful:

- o discover groups;
- o query group properties (e.g. related resource descriptions);
- o create a group;
- o remove a group;
- o add a group member;
- o remove a group member;
- o enumerate group members;
- o security and access control primitives.

In this proposal a (at least one) CoAP Proxy node is responsible for group membership management. A constrained node can specify which group it intends to join (or leave) using a CoAP request to the appropriate CoAP Proxy. To Join, the group name will be included in optional request header fields (explained below). These header

fields will be included in a PUT request to the Proxy. The Proxy-URI is set to the Group Management URI of the Proxy (found previously through the `"/.well-known/"` resource discovery mechanism). Note that in this solution also CoAP Proxies may exist in a network that are not capable of CoAP group operations.

Group names may be defined as arbitrary strings with a predefined maximum length (e.g. 268 characters or the maximum string length in a CoAP Option), or as URIs.

[ TBD: how can a client send a request to a group? Does it only need to know the group name (string or URI) or also an IP multicast address? One way is to send a CoAP request to the CoAP Proxy with a group URI directly in the Proxy-URI field. This avoids having to know anything related to IP multicast addresses. ]

This solution in principle supports both unreliable and reliable group communication. A client would indicate unreliable communication by sending a CoAP Non-Confirmable request to the CoAP Proxy, or reliable communication by sending a CoAP Confirmable request.

It is proposed that CoAP supports two Header Options for group "Join" and "Leave". These Options are Elective so they should be assigned an even number. Assuming the Type for "join" is `x` (value TBD), the Header Options are illustrated by the table in Figure 1:

| Type             | C/E | Name        | Data type | Length     | Default |
|------------------|-----|-------------|-----------|------------|---------|
| <code>x</code>   | E   | Group Join  | String    | 1-270<br>B | " "     |
| <code>x+2</code> | E   | Group Leave | String    | 1-270<br>B | " "     |

Figure 1: CoAP Header Options for Group Management

Figure 2 illustrates how a node can join or leave a group using the Header Options in a CoAP message:

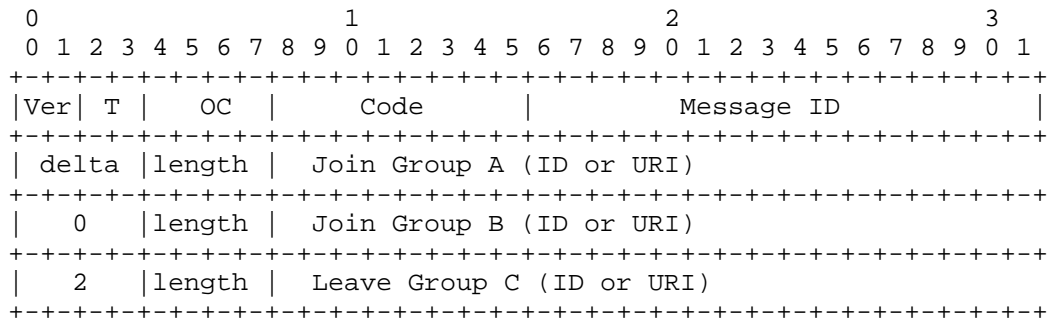


Figure 2: CoAP Message for Group Management

Header Fields for the above example:

Ver: 2-bit unsigned integer for CoAP Version. Set to 1 by implementation as defined by the CoAP specification.

T: 2-bit unsigned integer for CoAP Transaction Type. Either '0' Confirmation or '1' Non-Confirmable can be used for group "join" or "leave" request.

OC: 4-bit unsigned integer for Option Count. For this example, the value should be "3" since there are three option fields.

Code: 8-bit unsigned integer to indicate the Method in a Request or a Response Code in a Response message. Any Code can be used so the group management can be piggy-backed in either Request or Response message.

Message ID: 16-bit value assigned by the source to uniquely identify a pair of Request and Response.

CoAP defines a delta encoding for header options. The first delta is the "Type" for group join in this specific example. If the type for group join is x as illustrated in Figure 2, delta will be x. In the second header option, it is also a group join so the delta is 0. The third header option is a group leave so the delta is 2.

An alternative solution to using Header Options (explained above) is to use designated parameters in the query part of the URI in the Proxy-URI field of a POST (TBD: or PUT?) request to a Proxy's group management service resource advertised by DNS-SD. For example, to join group1 and leave group2:

```
coap://proxy1.bld2.example.com/groupmgt?j=group1&l=group2
```



## 6. DNS-SD Based Group Resource Manipulation

Ideally, all nodes in a given group (defined by its multicast IP address) must receive the same request with high probability. This will not be the case if there is diversity in the authority port (i.e. a diversity of dynamic port addresses across the group) or if the targeted resource is located at different paths on different nodes. Extending the definition of group membership to include port and path discovery is not desirable.

Therefore, some measures must be present to ensure uniformity in port number and resource name/location within a group.

A first solution in this respect is to couple groups to service descriptions in DNS (using DNS-SD as in [I-D.vanderstok-core-bc]). A service description for a multicast group may have a TXT record in DNS defining a schema X (e.g. "schema=DALI"), which defines by service standard X (e.g. "DALI") which resources a node supporting X MUST have. Therefore a multicast source can safely refer to all resources with corresponding operations as prescribed by standard X. For port numbers (which can be found using DNS-SD also) the same holds. Alternatively, only the default CoAP port may be used in all CoAP multicast requests.

## 7. Group Discovery and Member Discovery

CoAP defines a resource discovery capability, but does not yet specify how to discover groups (e.g. find a group to join or send a multicast message to) or to discover members of a group (e.g. to address selected group members by unicast). These topics are elaborated in more detail in [I-D.vanderstok-core-dna] including examples for using DNS-SD and CoRE Resource Directory.

### 7.1. DNS-SD

DNS-based Service Discovery [I-D.cheshire-dnsext-dns-sd] defines a conventional way to configure DNS PTR, SRV, and TXT records to enable enumeration of services, such as services offered by CoAP nodes, or enumeration of all CoAP nodes, within specified subdomains. A service is specified by a name of the form <Instance>.<ServiceType>.<Domain>, where the service type for CoAP nodes is \_coap.\_udp and the domain is a DNS domain name that identifies a group as in the examples above. For each CoAP end-point in a group, a PTR record with the name \_coap.\_udp and/or a PTR record with the name \_coap.\_udp.<Domain> is defined and it points to an SRV record having the <Instance>.<ServiceType>.<Domain> name.

All CoAP nodes in a given subdomain may be enumerated by sending a query for PTR records named `_coap._udp` to the authoritative DNS server for that zone. A list of SRV records is returned. Each SRV record contains the port and host name (AAAA record) of a CoAP node. The IP address of the node is obtained by resolving the host name. DNS-SD also specifies an optional TXT record, having the same name as the SRV record, which can contain "key=value" attributes. This can be used to store information about the device, e.g. `schema=DALI`, `type=switch`, `group=lighting.bldg6`, etc.

Another feature of DNS-SD is the ability to specify service sub-types using PTR records. For example, one could represent all the CoAP groups in a subdomain by PTR records with the name `_group._sub._coap._udp` or alternatively `_group._sub._coap._udp.<Domain>`.

## 7.2. CoRE Resource Directory

CoRE Resource Directory [I-D.shelby-core-resource-directory] defines the concept of a Resource Directory (RD) server where CoAP servers can register their resources offered and CoAP clients can discover these resources by querying the RD server. RD syntax can be mapped to DNS-SD syntax and vice versa [I-D.lynn-core-discovery-mapping], such that the above approach can be reused for group discovery and group member discovery.

Specifically, the Domain (d) parameter can be set to the group URI by an end-point registering to the RD. If an end-point wants to join multiple groups, it has to repeat the registration process for each group it wants to join.

## 8. Deployment Guidelines

### 8.1. Overview

We recommend to use IP multicast as the base solution for CoAP Group Communication, provided that the use case and network characteristics allow this. It has the advantage that it re-uses the IP multicast suite of protocols and can operate even if group members are distributed over both constrained and un-constrained network segments. Still, this approach may require specifying or implementing additional IP Multicast functionality in an LLN, in a backbone network, or in both - this will be evaluated in more detail in this section.

## 8.2. Implementation in Target Network Topologies

This section looks in more detail how an IP Multicast based solution can be deployed onto the various network topologies that we consider important for group communication use cases. Note that the chosen solution of IP Multicast for CoAP group communication works mostly independently from the underlying network topology and its specific IP multicast implementation.

Starting from the simplest case of a single LLN topology, we move to more complex topologies involving a backbone network or multiple LLNs. With "backbone" we refer here typically to a corporate LAN or VLAN, which constitutes a single broadcast domain by design. It could also be an in-home network. A multi-link backbone is also possible, if there is proper IP multicast routing or forwarding configured between these links. (The term 6LoWPAN Border Router or "6LBR" is used here for a border router, though our evaluation is not necessarily restricted to 6LoWPAN networks.)

### 8.2.1. Single LLN Topology

The simplest topology is a single LLN, where all the IP multicast source(s) and destinations are constrained nodes within this same LLN. Possible implementations of IP multicast routing and group administration for this topology are listed below.

#### 8.2.1.1. Mesh-Under Multicast Routing

The LLN may be set up in either a mesh-under or a route-over configuration. In the former case, the mesh routing protocol should take care of routing IP multicast messages throughout the LLN.

Because conceptually all nodes in the LLN are attached to a single link, there is in principle no need for nodes to announce their interest in multicast IP addresses via MLD (see Appendix A). A multicast message to a specific IP destination, which is delivered to all 6LoWPAN nodes by the mesh routing algorithm, is accepted by the IP network layer of that node only if it is listening on that specific multicast IP address and port.

#### 8.2.1.2. RPL Multicast Routing

The RPL routing protocol for LLNs provides support for routing to multicast IP destinations (Section 12 of [RFC6550]). Like regular unicast destinations, multicast destinations are advertised by nodes using RPL DAO messages. This functionality requires "Storing mode with multicast support" (Mode Of Operation, MOP is 3) in the RPL network.

Once all RPL routing tables in the network are populated, any RPL node can send packets to an IP multicast destination. The RPL protocol performs distribution of multicast packet both upward towards the DODAG root and downwards into the DODAG.

The text in Section 12 of the RPL specification clearly implies that IP multicast packets are distributed using link-layer unicast transmissions, looking at the use of the word "copied" in this section. Specifically in 6LoWPAN networks, this behavior conflicts with the requirement that IP multicast packets MUST be carried as link-layer 802.15.4 broadcast frames [RFC4944].

Assuming that link-layer unicast is indeed meant, this approach seems efficient only in a balanced, sparse tree network topology, or in situations where the fraction of nodes listening to a specific multicast IP address is low, or in duty cycled LLNs where link-layer broadcast is a very expensive operation.

#### 8.2.1.3. RPL Routers with Non-RPL Hosts

Now we consider the case that hosts exist in a RPL network that are not RPL-aware themselves, but rely on RPL routers for their IP connectivity beyond link-local scope. Note that the current RPL specification [RFC6550] leaves this case for future specification (see Section 16.4). Non-RPL hosts cannot advertise their IP multicast groups of interest via RPL DAO messages as defined above. Therefore in that case MLD could be used for such advertisements (State Change Report messages), with all or a subset of RPL routers acting in the role of MLD Routers as defined in [RFC3810]. However, as the MLD protocol is not designed specifically for LLNs it may be a burden for the constrained RPL router nodes to run the full MLD protocol. Alternatives are therefore proposed in Section 8.3.1.

#### 8.2.1.4. Trickle Multicast Forwarding

Trickle Multicast Forwarding [I-D.ietf-roll-trickle-mcast] is an IP multicast routing protocol suitable for LLNs, that uses the Trickle algorithm as a basis. It is a simple protocol in the sense that no topology maintenance is required. It can deal especially well with situations where the node density is a-priori unknown.

Nodes from anywhere in the LLN can be the multicast source, and nodes anywhere in the LLN can be multicast destinations.

Using Trickle Multicast Forwarding it is not required for IP multicast destinations (listeners) to announce their interest in a specific multicast IP address, e.g. by means of MLD. Instead, all multicast IP packets regardless of IP destination address are stored

and forwarded by all routers. Because forwarding is always done by multicast, both hosts and routers will be able to receive all multicast IP packets. Routers that receive multicast packets they are not interested in, will only buffer these for a limited time until retransmission can be stopped as specified by the protocol. Hosts that receive multicast packets they are not interested in, will discard multicast packets that are not of interest. Above properties seem to make Trickle especially efficient for cases where the multicast listener density is high and the number of distinct multicast groups relatively low.

#### 8.2.1.5. Other Route-Over Methods

Other known IP multicast routing methods may be used, for example flooding or other to be defined methods suitable for LLNs. An important design consideration here is whether multicast listeners need to advertise their interest in specific multicast addresses, or not. If they do, MLD is a possible option but also protocol-specific means (as in RPL) is an option. See Section 8.3.1 for more efficient substitutes for MLD targeted towards a LLN context.

#### 8.2.2. Single LLN with Backbone Topology

A LLN may be connected via a Border Router (e.g. 6LBR) to a backbone network, on which IP multicast listeners and/or sources may be present. This section analyzes cases in which IP multicast traffic needs to flow from/to the backbone, to/from the LLN.

##### 8.2.2.1. Mesh-Under Multicast Routing

Because in a mesh routing network conceptually all nodes in the LLN are attached to a single link, a multicast IP packet originating in the LLN is typically delivered by the mesh routing algorithm to the 6LBR as well, although there is no guaranteed delivery. The 6LBR may be configured to accept all IP multicast traffic from the LLN and then may forward such packets onto its backbone link. Alternatively, the 6LBR may act in an MLD Router or MLD Snooper role on its backbone link and decide whether to forward a multicast packet or not based on information learned from previous MLD Reports received on its backbone link.

Conversely, multicast packets originating on the backbone network will reach the 6LBR if either the backbone is a single link (LAN/VLAN) or IPv6 multicast routing is enabled on the backbone. Then, the 6LBR could simply forward all IP multicast traffic from the backbone onto the LLN. However, in practice this situation may lead to overload of the LLN caused by unnecessary multicast traffic. Therefore the 6LBR SHOULD only forward traffic that one or more nodes

in the LLN have expressed interest in, effectively filtering inbound LLN multicast traffic.

To realize this "filter", nodes on the LLN may use MLD to announce their interest in specific multicast IP addresses to the 6LBR. One option is for the 6LBR to act in an MLD Router role on its LLN interface. However, this may be too much of a "burden" for constrained nodes. Light-weight alternatives for MLD are discussed in Section 8.3.1.

#### 8.2.2.2. RPL Multicast Routing

For RPL routing within the 6LoWPAN, we first consider the case of an IP multicast source on the backbone network with one or more IP multicast listeners on the RPL LLN. Typically, the 6LBR would be the root of a DODAG so that the 6LBR can easily forward the IP multicast packet received on its backbone interface to the right RPL nodes in the LLN down along this DODAG (based on previously DAO-advertized destinations).

Second, a multicast source may be in the RPL LLN and listeners may be both on the LLN and on the backbone. For this case RPL defines that the multicast packet will propagate both up and down the DODAG, eventually reaching the DODAG root (typically a 6LBR) from which the packet can be routed onto the backbone in a manner specified in the previous section.

#### 8.2.2.3. RPL Routers with Non-RPL Hosts

For the case that a RPL LLN contains non-RPL hosts, the solutions from the previous section can be used if in addition RPL routers implement MLD or "MLD like" functionality similar to as described in Section 8.2.1.3.

#### 8.2.2.4. Trickle Multicast Forwarding

First, we consider the case of an IP multicast source node on the LLN (where all 6LRs support Trickle Multicast Forwarding) and IP multicast listeners that may be on the LLN and on the backbone. As Trickle will eventually deliver multicast packets also to a 6LBR, which acts as a Trickle Multicast router as well, the 6LBR can then forward onto the backbone in the ways described earlier in Section 8.2.2.1.

Second, for the case of an IP multicast source on the backbone and multicast listeners on both backbone and/or LLN, the 6LBR needs to forward multicast traffic from the backbone onto the LLN. Here, the

aforementioned problem (Section 8.2.2.1) of potentially overloading the LLN with unwanted backbone IP multicast traffic appears again.

A possible solution to this is (again) to let multicast listeners advertise their interest using MLD as described in Section 8.2.2.1 or to use an MLD alternative suitable for LLNs as described in Section 8.3.1. However, following this approach requires possibly an extension to Trickle Multicast Forwarding: the protocol should ensure that MLD-advertised information is somehow communicated to the 6LBR, possibly over multiple hops. MLD itself supports link-local communication only.

#### 8.2.2.5. Other Route-Over Methods

For other multicast routing methods used on the LLN, there are similar considerations to the ones in sections above: the strong need to filter IP multicast traffic coming into the LLN, the need for reporting multicast listener interest (e.g. with MLD or a to-be-defined MLD alternative) by constrained (6LoWPAN) nodes, and the need for LLN-internal routing as identified in the previous section such that the MLD communicated information can reach the 6LBR to be used there in multicast traffic filtering decisions.

#### 8.2.3. Multiple LLNs with Backbone Topology

Now the case of a single backbone network with two or more LLNs attached to it via 6LBRs is considered. For this case all the considerations and solutions of the previous section can be applied.

For the specific case that a source on a backbone network has to send to a very large number of destination located on many LLNs, the use of IGMP/MLD Proxying [RFC4605] with a leaf IGMP/MLD Proxy located in each 6LBR may be useful. This method only is defined for a tree topology backbone network with the IP multicast source at the root of the tree.

#### 8.2.4. LLN(s) with Multiple 6LBRs

[ TBD: an LLN with multiple 6LBRs may require some additional consideration. Any need to synchronize mutually on multicast listener information? ]

#### 8.2.5. Conclusions

For all network topologies that were evaluated, CoAP group communication can be in principle supported with IP Multicast, making use of existing protocols. For the case of Trickle Multicast Forwarding, it appears that an addition to the protocol is required

such that information about multicast listeners can be distributed towards the 6LBR. Opportunities were identified for an "MLD-like" or "MLD-lightweight" protocol specifically suitable for LLNs, which should inter-work with regular MLD on the backbone network. Such MLD variants are further analyzed in Section 8.3.1.

### 8.3. Implementation Considerations

In this section various implementation aspects are considered such as required protocol implementations, additional functionality of the 6LBR and backbone network equipment.

#### 8.3.1. MLD Implementation on LLNs and MLD alternatives

In previous sections, it was mentioned that the MLDv2 protocol [RFC3810] may be too costly for use in a LLN. MLD relies on periodic link-local multicast operations to maintain state. Also it is optimized to fairly dynamic situations where multicast listeners may come and go over time. Such dynamic situations are less frequently found in typical LLN use cases such as building control, where multicast group membership can remain constant over longer periods of time (e.g. months) after commissioning.

Hence, a viable strategy is to implement a subset of MLD functionality in 6LoWPAN nodes which is just enough for the required functionality. A first option is that 6LoWPAN Routers, like MLD Snooters, passively listen to MLD State Change Report messages and handle the learned ("snooped") IP multicast destinations in the way defined by the multicast routing protocol they are running (e.g. for RPL, Routers advertise these destinations using DAO messages).

A second option is to use MLD as-is but adapt the recommended parameter values such that operation on a LLN becomes more efficient. [RFC6636] could be a guideline in this case.

A third option is to standardize a new protocol, taking a subset of MLD functionality into a "MLD for 6LoWPAN" protocol to support constrained nodes optimally.

A fourth option is now presented, which seems attractive in that it minimizes standardization, implementation and network communication overhead all at the same time. This option is to specify a new Multicast Listener Option (MLO) as an addition to the 6LoWPAN-ND [RFC6775] protocol communication that is anyway ongoing between a 6LoWPAN host and router(s). This MLO is preferably designed to be maximally similar to the Address Registration Option (ARO), which minimizes the need for additional program code on constrained nodes. With an MLO, instead of registering a host's unicast IP address as



with ARO, a host "registers" its interest in a multicast IPv6 address. Unlike the ARO, multiple MLO can be used in the same ND packet. A registration period is also defined in the MLO just like in the ARO. MLO allows a host to persistently register as a listener to IP multicast traffic and to avoid the overhead of periodic multicast communication which is required for the regular MLD protocol.

[ TBD: consider what aspects are needed/not needed for CoAP/LLN applications. Will MLDv1 suffice? What to do with options like 'source specific' and include/exclude. Source-specific can also be dealt with at the destination host by filtering? Do we need limits on number of records per packet? Do we need a higher MLD reliability setting - see the parameters in the MLD RFC ]

#### 8.3.2. 6LBR Implementation

To support mixed backbone/LLN scenarios in CoAP group communication, it is RECOMMENDED that a 6LowPAN Border Router (6LBR) will act in an MLD Router role on the backbone link. If this is not possible then the 6LBR SHOULD be configured to act as an MLD Multicast Address Listener and/or MLD Snooper on the backbone link.

#### 8.3.3. Backbone IP Multicast Infrastructure

For corporate/professional applications, most routing and switching equipment that is currently on the market is IPv6 capable. For that reason backbone infrastructure operating IPv4 only is considered out of scope in this document, at least for the backbone network segment(s) where IP multicast destinations are present. What is still in scope is for example an IPv4-only HTTP client that wants to send a group communication message via a HTTP-CoAP proxy as considered in [I-D.castellani-core-advanced-http-mapping].

The availability of, and requirements for, IP multicast support may depend on the specific installation use case. For example, the following cases may be relevant for new IP based building control installations:

1. System deployed on existing IP (Ethernet/WiFi/...) infrastructure, shared with existing IP devices (PCs)
2. Newly designed and deployed IP (Ethernet/WiFi/...) infrastructure, to be shared with other IP devices (PCs)
3. Newly designed and deployed IP (Ethernet/WiFi/...) infrastructure, exclusively used for building control.

Besides physical separation the building control backbone can be separated from regular (PC) infrastructure by using a different VLAN. A typical corporate installation will have many LAN switches and/or routing switches, which pass through IP multicast traffic but on the other hand do not support acting in the Router role of MLD/IGMP. Perhaps for case 2) and 3) above it is acceptable to add a MLD/IGMP capable router somewhere in the network, while for case 1) this may not be the case.

[TBD: consider the influence of WiFi based backbone networks. What if 6LBRs are at the same time also WiFi routers? What if 6LBRs have an Ethernet connection to legacy WiFi routers? Check if equivalent with Ethernet backbone.]

## 9. Miscellaneous Topics

This section collects miscellaneous text, topics or proposals related to CoAP group communication which do not directly fit into any of the preceding sections.

### 9.1. CoAP Multicast and HTTP Unicast Interworking

CoAP supports operation over UDP multicast, while HTTP does not. For use cases where it is required that CoAP group communication is initiated from an HTTP end-point, it would be advantageous if the HTTP-CoAP Proxy supports mapping of HTTP unicast to CoAP group communication based on IP multicast. One possible way of operation of such HTTP-CoAP Proxy is illustrated in Figure 3. Note that this topic is covered in more detail in [I-D.castellani-core-advanced-http-mapping].

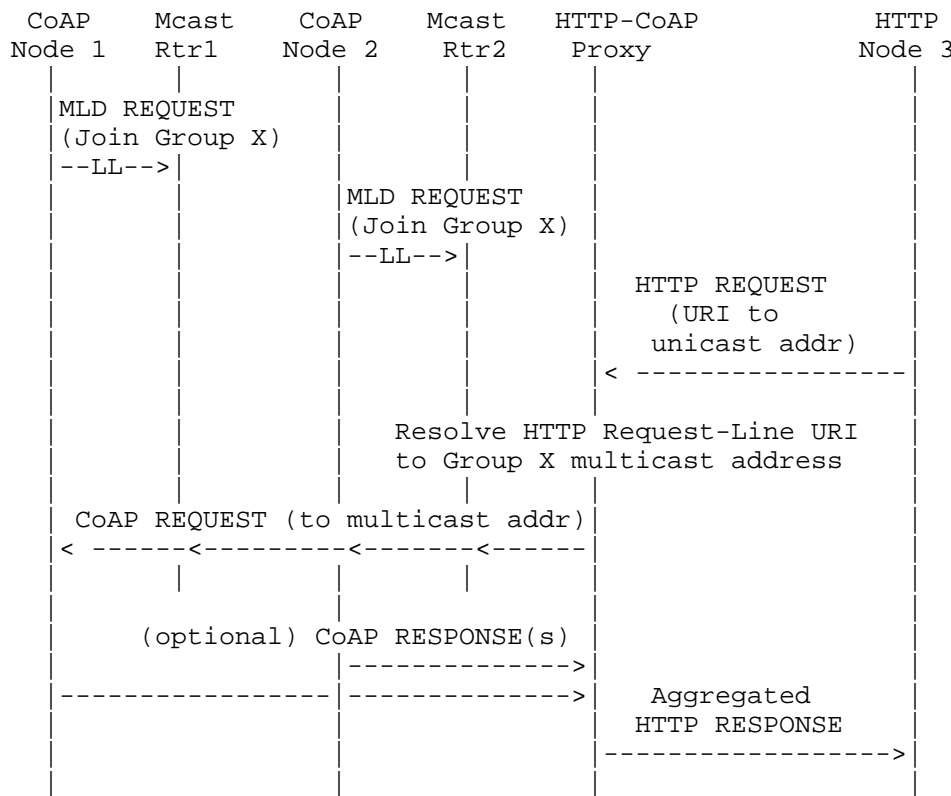


Figure 3: CoAP Multicast and HTTP Unicast Interworking

Note that Figure 3 illustrates the case of IP multicast as the underlying group communications mechanism. MLD denotes the Multicast Listener Discovery protocol ([RFC3810], Appendix A) and LL denotes a Link-Local multicast.

A key point in Figure 3 is that the incoming HTTP Request (from node 3) will carry a Host request-header field that resolves in the general Internet to the proxy node. At the proxy node, this hostname and/or the Request-Line URI will then possibly be mapped (as detailed in [I-D.castellani-core-advanced-http-mapping]) and again resolved (with the CoAP scheme) to an IP multicast address. This may be accomplished, for example, by using DNS or DNS-SD (Section 7). The proxy node will then IP multicast the CoAP Request (corresponding to the received HTTP Request) via multicast routers to the appropriate nodes (i.e. nodes 1 and 2).

In terms of the HTTP Response, Figure 3 illustrates that it will be generated by the proxy node based on aggregated responses of the CoAP nodes and sent back to the client in the general Internet that sent the HTTP Request (i.e. node 1). In [I-D.castellani-core-advanced-http-mapping] the HTTP Response that the Proxy may use to aggregate multiple CoAP responses is described in more detail. So in terms of overall operation, the CoAP proxy can be considered to be a "non-transparent" proxy according to [RFC2616]. Specifically, [RFC2616] states that a "non-transparent proxy is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction or anonymity filtering."

An alternative to the above is using a Forward Proxy. In this case, the CoAP request URI is carried in the HTTP Request-Line (as defined in [I-D.ietf-core-coap] Section 10.2) in a HTTP request sent to the IP address of the Proxy.

#### 10. Acknowledgements

Thanks to all CoRE WG members who participated in the IETF 82 discussions, which was the trigger to initiate this document.

#### 11. IANA Considerations

This memo includes no request to IANA.

#### 12. Security Considerations

The basic security aspects of group communication for CoAP are discussed in [I-D.ietf-core-groupcomm]. The DICE I-D [I-D.keoh-dice-multicast-security] takes as input many of the security requirements listed in Section 4.3 for proposing added DTLS protection for CoAP group communication.

Some additional considerations for group communication security can be found in [RFC7258] which warns of the dangers of pervasive monitoring. CoAP group communication solutions built on IP multicast should pay particular heed to these dangers. This is because IP multicast is easier to intercept (e.g. and to secretly record) compared to unicast traffic. Also, CoAP traffic is meant for the Internet of Things. This means that CoAP traffic is often used for the control and monitoring of critical infrastructure (e.g. lights, alarms, etc.)

For example, an attacker may want to record all the CoAP traffic going over the smart grid (electrical utility) of a country and try

to determine critical control nodes. CoAP multicast traffic is inherently more vulnerable (compared to a unicast packet) as the same packet will be replicated over many links so there is a much higher probability of it getting captured by a pervasive monitoring system. Even if all the CoAP multicast traffic is protected via DTLS ([I-D.keoh-dice-multicast-security]), an attacker may attempt to capture the traffic and perform an off-line attack. Though of course having the multicast traffic protected is always desirable as it significantly raises the cost to an attacker (e.g. to break the encryption) versus unprotected multicast traffic.

### 13. References

#### 13.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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, June 1999.
- [RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, March 2004.
- [RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.
- [RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, April 2005.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.
- [RFC4605] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")", RFC 4605, August 2006.

- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5374] Weis, B., Gross, G., and D. Ignjatic, "Multicast Extensions to the Security Architecture for the Internet Protocol", RFC 5374, November 2008.
- [RFC5867] Martocci, J., De Mil, P., Riou, N., and W. Vermeulen, "Building Automation Routing Requirements in Low-Power and Lossy Networks", RFC 5867, June 2010.
- [RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012.
- [RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, May 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, May 2014.

### 13.2. Informative References

- [Banerjee01] Banerjee, B. and B. Bhattacharjee, "A Comparative Study of Application Layer Multicast Protocols", 2001, <<http://wmedia.grnet.gr/P2PBackground/a-comparative-study-ofALM.pdf>>.
- [I-D.castellani-core-advanced-http-mapping] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for HTTP-CoAP Mapping Implementations", draft-castellani-core-advanced-http-mapping-03 (work in progress), December 2013.

- [I-D.cheshire-dnsext-dns-sd]  
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-11 (work in progress), December 2011.
- [I-D.eggert-core-congestion-control]  
Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [I-D.ietf-6lowpan-routing-requirements]  
Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem Statement and Requirements for 6LoWPAN Routing", draft-ietf-6lowpan-routing-requirements-10 (work in progress), November 2011.
- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-14 (work in progress), October 2013.
- [I-D.ietf-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-18 (work in progress), December 2013.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-13 (work in progress), April 2014.
- [I-D.ietf-roll-trickle-mcast]  
Hui, J. and R. Kelsey, "Multicast Protocol for Low power and Lossy Networks (MPL)", draft-ietf-roll-trickle-mcast-09 (work in progress), April 2014.
- [I-D.keoh-dice-multicast-security]  
Keoh, S., Kumar, S., Garcia-Morchon, O., Dijk, E., and A. Rahman, "DTLS-based Multicast Security in Constrained Environments", draft-keoh-dice-multicast-security-07 (work in progress), May 2014.
- [I-D.lynn-core-discovery-mapping]  
Lynn, K. and Z. Shelby, "CoRE Link-Format to DNS-Based Service Discovery Mapping", draft-lynn-core-discovery-mapping-02 (work in progress), October 2012.

- [I-D.rahman-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP",  
draft-rahman-core-groupcomm-07 (work in progress), October  
2011.
- [I-D.shelby-core-coap-req]  
Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R.  
Kelsey, "CoAP Requirements and Features", draft-shelby-  
core-coap-req-02 (work in progress), October 2010.
- [I-D.shelby-core-resource-directory]  
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource  
Directory", draft-shelby-core-resource-directory-05 (work  
in progress), February 2013.
- [I-D.vanderstok-core-bc]  
Stok, P. and K. Lynn, "CoAP Utilization for Building  
Control", draft-vanderstok-core-bc-05 (work in progress),  
October 2011.
- [I-D.vanderstok-core-dna]  
Stok, P., Lynn, K., and A. Brandt, "CoRE Discovery,  
Naming, and Addressing", draft-vanderstok-core-dna-02  
(work in progress), July 2012.
- [Lao05]    Lao, L., Cui, J., Gerla, M., and D. Maggiorini, "A  
Comparative Study of Multicast Protocols: Top, Bottom, or  
In the Middle?", 2005,  
<[http://www.cs.ucla.edu/NRL/hpi/AggMC/papers/  
comparison\\_gi\\_2005.pdf](http://www.cs.ucla.edu/NRL/hpi/AggMC/papers/comparison_gi_2005.pdf)>.

#### Appendix A. Multicast Listener Discovery (MLD)

In order to extend the scope of IP multicast beyond link-local scope, an IP multicast routing protocol has to be active in routers on an LLN. To achieve efficient multicast routing (i.e. avoid always flooding multicast IP packets), routers have to learn which hosts need to receive packets addressed to specific IP multicast destinations.

The Multicast Listener Discovery (MLD) protocol [RFC3810] (or its IPv4 pendant IGMP) is today the method of choice used by an (IP multicast enabled) router to discover the presence of multicast listeners on directly attached links, and to discover which multicast addresses are of interest to those listening nodes. MLD was specifically designed to cope with fairly dynamic situations in which multicast listeners may join and leave at any time.



IGMP/MLD Snooping is a technique implemented in some corporate LAN routing/switching devices. An MLD snooping switch listens to MLD State Change Report messages from MLD listeners on attached links. Based on this, the switch learns on what LAN segments there is interest for what IP multicast traffic. If the switch receives at some point an IP multicast packet, it uses the stored information to decide onto which LAN segment(s) to send the packet. This improves network efficiency compared to the regular behavior of forwarding every incoming multicast packet onto all LAN segments. An MLD snooping switch may also send out MLD Query messages (which is normally done by a device in MLD Router role) if no MLD Router is present.

[RFC6636] discusses optimal tuning of the parameters of MLD for routers for mobile and wireless networks. These guidelines may be useful when implementing MLD in LLNs.

#### Appendix B. CoAP-Observe Alternative to Group Communication

The CoAP Observation extension [I-D.ietf-core-observe] can be used as a simple (but very limited) alternative for group communication. A group in this case consists of a CoAP server hosting a specific resource, plus all CoAP clients observing that resource. The server is the only group member that can send a group message. It does this by modifying the state of a resource under observation and subsequently notifying its observers of the change. Serial unicast is used for sending the notifications. This approach can be a simple alternative for networks where IP multicast is not available or too expensive.

The CoAP-Observe approach is unreliable in the sense that, even though Confirmable CoAP messages may be used, there are no guarantees that an update will be received. For example, a client may believe it is observing a resource while in reality the server rebooted and lost its listener state.

#### Authors' Addresses

Esko Dijk (editor)  
Philips Research

Email: [esko.dijk@philips.com](mailto:esko.dijk@philips.com)

Akbar Rahman (editor)  
InterDigital Communications, LLC

Email: [Akbar.Rahman@InterDigital.com](mailto:Akbar.Rahman@InterDigital.com)

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: January 10, 2013

T. Fossati  
KoanLogic  
S. Loreto  
Ericsson  
July 9, 2012

Resource Discovery through Proxies  
draft-fossati-core-fp-link-format-attribute-00

Abstract

The aim of this draft is to open a discussion on how to make it possible to advertise the fact that a given resource hosted by a server can only be reached through a specific CoAP Proxy.

This memo proposes the definition of the "fp" (forward proxy) CoAP link format attribute, that can be used to inform CoAP endpoints that a given resource can be reached by passing through the advertising Proxy.

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 January 10, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|   |   |
|---|---|
| 1. Introduction . . . . .                 | 3 |
| 1.1. Requirements Language . . . . .      | 3 |
| 2. Proxied Discovery Scenario . . . . .   | 3 |
| 3. The fp Link Format Attribute . . . . . | 4 |
| 4. IANA Considerations . . . . .          | 5 |
| 4.1. fp Attribute . . . . .               | 5 |
| 5. Security Considerations . . . . .      | 5 |
| 6. References . . . . .                   | 6 |
| 6.1. Normative References . . . . .       | 6 |
| 6.2. Informative References . . . . .     | 6 |
| Authors' Addresses . . . . .              | 6 |

## 1. Introduction

The discovery mechanism described in [I-D.ietf-core-link-format] assumes cheap and pervasive multicast. However as discussed in [I-D.shelby-core-resource-directory] direct discovery of resources is not always practical due to limitations in the underlying radio link (see Section 1 of [I-D.ietf-6lowpan-nd]), the absence of a multicast routing protocol to bridge through different links, sleeping nodes, disperse networks.

The Resource Directory (RD) provides a first solution hosting descriptions of resources held on other servers and allowing lookups to be performed for those resources. The current solution however does not address the scenario where the URI (of the resource of interest) is associated to a CoAP origin server that can only be accessed through a CoAP proxies either for topological and/or security reasons or because it is a sleepy origin server.

Given their topological role, CoAP Proxies (Section 5.7 of [I-D.ietf-core-coap]) can be used effectively to address the above mentioned scenarios. However, in order to achieve this capability, the fact that a given resource is made available through a proxy must be made explicit to consuming endpoints, so that they can use the Proxy-Uri Option to dereference the final target.

This memo defines the "fp" (forward proxy) CoAP link format attribute, that can be used to inform CoAP endpoints that a given resource can be reached by passing through the advertising Proxy.

### 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].

## 2. Proxied Discovery Scenario

Consider the scenario depicted in Figure 1. Two separate CoAP links are proxied by P. Node A hosts resource /res of type "x", and P knows it -- either through explicit or implicit mechanism (e.g. previous discovery on the local link, or a co-located RD, etc.)

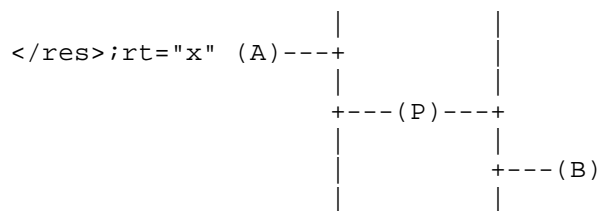


Figure 1

We would like to allow B to discover that A hosts a resource with type "x" even if A can't be directly reached by B.

P may in principle let this information filter from one link to the other, but given the mechanisms currently defined for the discovery via /.well-known/core (Section 2.1 of [I-D.ietf-core-link-format]), there is no way for a consuming node to ascertain that an advertised link is to be accessed through a given forward Proxy or by a direct route.

We may choose to use the anchor parameter in the link and define a new relation name to express the "proxied by" relation, but this may actually have zero chance to succeed because of the freedom left to a consuming node to actually ignore anchored links (Section 2.3 of [I-D.ietf-core-link-format]).

### 3. The fp Link Format Attribute

The proposed solution, instead, envisages a new link format attribute, "fp" that is added by the Proxy to the original set of attributes of the linked resource to inform the requesting endpoint that the advertised (absolute) URI must be requested to the advertising Proxy using the Proxy-URI Option, as illustrated in Figure 2. The "fp" link format attribute MAY be set to the Proxy IP address.

When advertised on a link different from the one on which it resides, the original resource link SHALL be transformed by the Proxy into an absolute URI that can be used as-is in a Proxy-Uri Option by the requesting node.

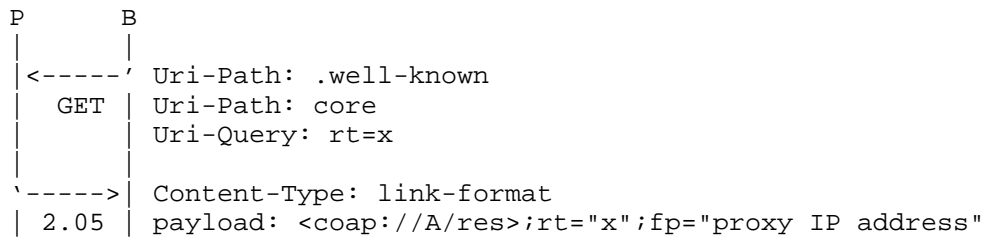
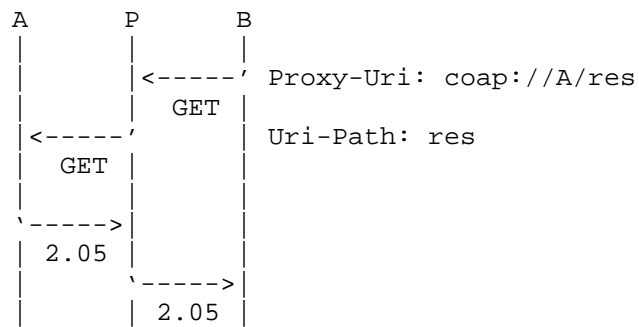


Figure 2

Note that in case the "fp" attribute is present, the URI-Reference in the link-value [RFC5988] MUST always be a URI and not a relative-ref [RFC3986].

The forwarding path to /res is now set up, and B can reach it through P using the Proxy-Uri Options as follows:



## 4. IANA Considerations

### 4.1. fp Attribute

This section defines a new Web Linking [RFC5988] attribute for use with [I-D.ietf-core-link-format]. The "fp" (forward proxy) CoAP link format attribute, that can be used by Proxy nodes to inform CoAP endpoints that a given resource can be reached by passing through the advertising Proxy.

## 5. Security Considerations

The mechanism specified in this document shares the same security concerns as the discovery process described in [I-D.ietf-core-link-format].

Especially critical to the CoAP network consistency, is the fact that in NoSec mode a malicious attacker could poison the response of a query to the /.well-known/core in order to re-route traffic.

## 6. References

### 6.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format",  
draft-ietf-core-link-format-13 (work in progress),  
May 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform  
Resource Identifier (URI): Generic Syntax", STD 66,  
RFC 3986, January 2005.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

### 6.2. Informative References

- [I-D.ietf-6lowpan-nd]  
Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor  
Discovery Optimization for Low Power and Lossy Networks  
(6LoWPAN)", draft-ietf-6lowpan-nd-18 (work in progress),  
October 2011.
- [I-D.shelby-core-resource-directory]  
Krcro, S. and Z. Shelby, "CoRE Resource Directory",  
draft-shelby-core-resource-directory-02 (work in  
progress), October 2011.

Authors' Addresses

Thomas Fossati  
KoanLogic  
Via di Sabbiano, 11/5  
Bologna 40100  
Italy

Email: [tho@koanlogic.com](mailto:tho@koanlogic.com)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: [salvatore.loreto@ericsson.com](mailto:salvatore.loreto@ericsson.com)





Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: January 10, 2013

T. Fossati  
KoanLogic  
P. Giacomin  
Freelance  
S. Loreto  
Ericsson  
July 9, 2012

Monitor Option for CoAP  
draft-fossati-core-monitor-option-00

Abstract

This memo defines Monitor, an additional Option for the Constrained Application Protocol (CoAP) especially targeted at sleepy sensors.

The Monitor Option complements the typical Observe pattern, enabling the tracking of a resource hosted by a node sleeping most of the time, by taking care of establishing and maintaining an Observe relationship with the (sleepy) origin on behalf of the (sleepy) client.

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 January 10, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|   |   |
|---|---|
| 1. Introduction                           | 3 |
| 1.1. Requirements Language and Motivation | 3 |
| 2. Monitor Option                         | 3 |
| 2.1. Public Monitor Registration          | 4 |
| 2.2. Monitor De-registration              | 6 |
| 2.2.1. Explicit De-registration           | 6 |
| 2.2.2. Implicit De-registration           | 6 |
| 2.3. Resource Refresh                     | 6 |
| 3. Acknowledgements                       | 7 |
| 4. IANA Considerations                    | 7 |
| 5. Security Considerations                | 7 |
| 6. Normative References                   | 7 |
| Authors' Addresses                        | 8 |

## 1. Introduction

The proposal described in this memo covers the following use case: a node N, which is sleeping most of the time, depends on one or more resources hosted at another sleepy node M. In cases as such, the probability of an empty intersection between their respective wake periods is very high, making it hard for the two to synchronize.

In this scenario, using the basic observe [I-D.ietf-core-observe] functionality is not enough, as it could lead to lost state updates in case N is offline while M pushes its notifications; further, the observation may never bootstrap since its initialization needs both client and origin awake at the same time.

This memo introduces an extension to the Proxy caching functionality that give the Proxy an explicit mediation role in the sleepy-to-sleepy CoAP [I-D.ietf-core-coap] communication.

### 1.1. Requirements Language and Motivation

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].

This specification makes use of the following terminology:

**Sleepy Device:** a sensor/actuator (usually battery operated) that powers down its radio beyond the normal radio duty cycle in order to save energy.

and tries to provide an in-protocol solution for requirement REQ3 in [I-D.shelby-core-coap-req]:

The ability to deal with sleeping nodes. Devices may be powered down at any point in time but periodically "wake up" for brief periods of time.

## 2. Monitor Option

|         |          |         |         |         |         |
|---------|----------|---------|---------|---------|---------|
| +-----+ | +-----+  | +-----+ | +-----+ | +-----+ | +-----+ |
| No.     | C/E      | Name    | Format  | Length  | Default |
| +-----+ | +-----+  | +-----+ | +-----+ | +-----+ | +-----+ |
| XX      | Critical | Monitor | (none)  | 0 B     | (none)  |
| +-----+ | +-----+  | +-----+ | +-----+ | +-----+ | +-----+ |

The Monitor Option is a variant of the Observe Option that is aimed at solving some issues that may occur when sleepy sensors are

involved.

Suppose that the resource of interest is not cached anywhere, and a sleepy endpoint wants to Observe it through a Proxy. If the origin of the requested resource is sleeping at the time the observation is requested, the requesting node gets an error, and may need to stay awake and retry until the target node gets ready -- which is clearly not an option in case the sensor has a very small duty cycle.

The Monitor Option is used to ask a Proxy to keep a given resource fresh by observing it, while the requesting node is sleeping. Thus the sleepy sensor can possibly get the latest representation published by the monitored resource when it wakes up, even if the origin is sleeping -- and was sleeping at the time the Monitor has been requested.

The Monitor Option is critical and **MUST** be present in the request only. If the Proxy does not recognize it, a 4.02 (Bad Option) **MUST** be returned to the client.

### 2.1. Public Monitor Registration

|   |         |   |
|---|---------|---|
| P |         | C |
|   | POST    |   |
|   | <-----> |   |
|   |         |   |
|   | 2.01    |   |
|   | +-----> |   |
|   |         |   |
|   |         |   |

```

Proxy-URI: coap://sleepy.example.org/res
Monitor: <empty>
Max-Age: 86400
Content-Type: application/json
Location-Path: temp
Location-Path: res

```

Figure 1

The client POST's the resource to be monitored, identified by the Proxy-URI. The request message contains an empty Monitor Option, and possibly specifies a TTL (i.e. an implicit de-registration indication) for the monitor through Max-Age. One or more content types for the acceptable representations of the resource are optionally specified via the Accept option. In case no TTL is supplied, a default value of 3600 seconds is assumed.

The operation creates a "monitor" resource at the Proxy, that **MUST** maintain a fresh carbon copy of one or more representations of the requested resource depending on the supplied Content-Type. For convenience, multiple "monitor" resources corresponding to the same target resource, can be coalesced into the same monitor object at the Proxy -- possibly with the same URI. In such case, a set containing

one entry for each registered client is kept, which holds the client identities, their expiry and one or more preferred media types for their representation(s). When all entries are deleted (either because clients have explicitly deregistered the monitor, or the monitor period has expired), the corresponding "monitor" object is deleted. Note that an underlying cache entry MAY still be kept in case the cached representation(s) are still fresh (i.e. the Max-Age of the "monitor" resource and Max-Age of the target resource have completely different semantics.)

If the monitor resource is successfully created, the server MUST return a 2.01 response containing one or more Location-Path and/or Location-Query Options to identify the monitored resource instance, which can be used from now on by the requester as an alias to the target resource.

At a later time, the client wakes up and wants to access the monitored resource. It does so by requesting the Proxy monitor resource that has been previously created.

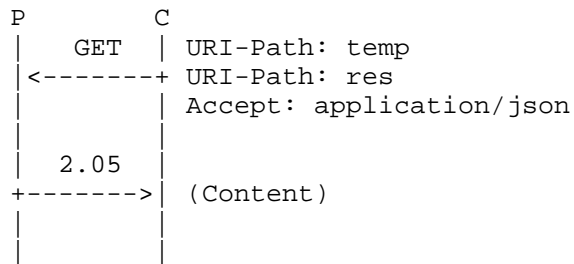


Figure 2

In case the observation on the target node has not been started because the Proxy has not yet been able to contact the origin, the Proxy will return a [TBD] error code.

In case the requested resource was not present on the origin, the Proxy will return an empty response (i.e. one with no payload.)

[[XXX: add an explicit response code perhaps like HTTP 204 ?]]

In case the monitor resource is not found in the Proxy, either because the Proxy has rebooted and lost its state, or the monitor resource has been de-registered (see Section 2.2), a 4.04 response code is returned to the client -- that can recreate it, if needed.

## 2.2. Monitor De-registration

The monitor object **MUST** be deleted at the Proxy when all its associated resources have been de-registered or have expired.

In order to save storage, a Proxy **MAY** decide to delete a monitor resource in case it has not been requested for a sufficiently long time, or for any other reason. Note that the Proxy may also reboot and lose its state, including the state associated to any monitored resource. The requester can realize that the state at the Proxy has been lost, and re-instantiate the monitor, when it receives an unexpected 4.04 from the "monitor" resource.

### 2.2.1. Explicit De-registration

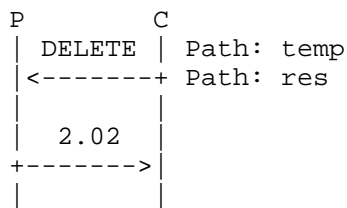


Figure 3

Explicit de-registration is performed by a client, with a DELETE on the URI returned by the Proxy on the corresponding registration.

### 2.2.2. Implicit De-registration

Implicit de-registration **MUST** occur when the monitoring period specified by the client via Max-Age expires. If no Max-Age was supplied at registration time, a default of 3600 seconds **MUST** be assumed.

## 2.3. Resource Refresh

In order to minimize the number of messages used by the monitoring process, the Proxy **MUST** try to install an observation on the requested resource. In case this first attempt fails, the Proxy **MAY** fall back to repeated poll whose duration is upper bounded by the Max-Age value indicated by the client during registration.

Usual cache validation **MUST** be applied to the cached copy of the monitored resource.

### 3. Acknowledgements

Bruce Nordman and Matthieu Vial for discussing and giving advice on some of the ideas contained in this document.

### 4. IANA Considerations

The following entries are added to the CoAP Option Numbers registry:

| Number | Name    | Reference |
|--------|---------|-----------|
| 2m+1   | Monitor | RFC XXXX  |

### 5. Security Considerations

Threat: cache poisoning.  
Countermeasure: authenticate sender.

Threat: unauthorized de-registration  
Countermeasure: authenticate requester.

Threat: Proxy resources' exhaustion.  
Countermeasure: authenticate requester + quota limit.

Threat: global state loss.  
Countermeasure: cache redundancy.

Threat: DoS on remote constrained resource via unneeded monitoring.  
Countermeasure: access control on the constrained resource (?)

### 6. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-05 (work in progress), March 2012.
- [I-D.shelby-core-coap-req]  
Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R.



Kelsey, "CoAP Requirements and Features",  
draft-shelby-core-coap-req-04 (work in progress),  
May 2011.

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

#### Authors' Addresses

Thomas Fossati  
KoanLogic  
Via di Sabbbiuno, 11/5  
Bologna 40100  
Italy

Email: tho@koanlogic.com

Pierpaolo Giacomini  
Freelance

Email: yrz@anche.no

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: salvatore.loreto@ericsson.com



CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: December 10, 2018

C. Bormann  
Universitaet Bremen TZI  
June 08, 2018

Multipart Content-Format for CoAP  
draft-fossati-core-multipart-ct-05

Abstract

This memo defines application/multipart-core, an application-independent media-type that can be used to combine representations of several different media types into a single CoAP message-body with minimal framing overhead, each along with a CoAP Content-Format identifier.

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 <https://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 December 10, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

|  |   |
|--|---|
| 1. Introduction . . . . .  | 2 |
| 2. Multipart Content-Format Encoding . . . . .   | 2 |
| 3. IANA Considerations . . . . .   | 3 |
| 3.1. Registration of media type application/multipart-core . .                               | 3 |
| 3.2. Registration of a Content-Format identifier for<br>application/multipart-core . . . . . | 4 |
| 4. Security Considerations . . . . .   | 4 |
| 5. References . . . . .  | 5 |
| 5.1. Normative References . . . . .  | 5 |
| 5.2. Informative References . . . . .  | 5 |
| Acknowledgements . . . . .   | 5 |
| Author's Address . . . . .   | 5 |

## 1. Introduction

This memo defines application/multipart-core, an application-independent media-type that can be used to combine representations of several different media types into a single CoAP [RFC7252] message-body with minimal framing overhead, each along with a CoAP Content-Format identifier.

This simple and efficient binary framing mechanism can be employed to create application specific request and response bodies which build on multiple already existing media types.

Applications using the application/multipart-core Content-Format define the internal structure of the application/multipart-core representation.

For example, one way to structure the sub-types specific to an application/multipart-core container is to always include them at the same fixed position. This specification allows to indicate that an optional part is not present by substituting a null value for the representation of the part.

Optionally, an application might use the general format defined here, but also register a new media type and an associated Content-Format identifier -- typically one in the range 10000-64999 -- instead of using application/multipart-core.

## 2. Multipart Content-Format Encoding

A representation of media-type application/multipart-core contains a collection of zero or more representations, each along with their respective content format.

The collection is encoded as a CBOR [RFC7049] array with an even number of elements. The second, fourth, sixth, etc. element is a byte string containing a representation, or the value "null" if an optional part is indicated as not given. The first, third, fifth, etc. element is an unsigned integer specifying the content format ID of the representation following it. Where needed by an application, each such format identifying element alternatively can be a text string giving the media type name plus potentially some parameters.

For example, a collection containing two representations, one with content format ID 42 and one with content format ID 0, looks like this in CBOR diagnostic notation:

```
[42, h'0123456789abcdef', 0, h'3031323334']
```

For illustration, the structure of an application/multipart-core representation can be described by the CDDL [I-D.ietf-cbor-cddl] specification in Figure 1:

```
multipart-core = [* multipart-part]
multipart-part = (type: uint .size 2 / text, part: bytes / null)
```

Figure 1: CDDL for application/multipart-core

### 3. IANA Considerations

#### 3.1. Registration of media type application/multipart-core

IANA is requested to register the following media type [RFC6838]:

Type name: application

Subtype name: multipart-core

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations Section of RFCthis

Interoperability considerations: N/A

Published specification: RFCthis

Applications that use this media type: Applications that need to combine representations of potentially several media types into one, e.g., EST-CoAP [I-D.ietf-ace-coap-est]

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): N/A

File extension(s): N/A

Macintosh file type code(s): N/A

Person & email address to contact for further information:  
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: CoRE WG

Change controller: IESG

Provisional registration? (standards tree only): no

### 3.2. Registration of a Content-Format identifier for application/multipart-core

IANA is requested to register the following Content-Format to the "CoAP Content-Formats" subregistry, within the "Constrained RESTful Environments (CoRE) Parameters" registry, from the IETF Review space (specifically, 256..999):

| Media Type                 | Encoding | ID   | Reference |
|----------------------------|----------|------|-----------|
| application/multipart-core | --       | TBD1 | RFCthis   |

## 4. Security Considerations

The security considerations of [RFC7049] apply. In particular, resource exhaustion attacks may employ large values for the byte

string size fields, or deeply nested structures of recursively embedded application/multipart-core representations.

## 5. References

### 5.1. Normative References

- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

### 5.2. Informative References

- [I-D.ietf-ace-coap-est] Stok, P., Kampanakis, P., Kumar, S., Richardson, M., Furuher, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-ietf-ace-coap-est-01 (work in progress), June 2018.
- [I-D.ietf-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.

## Acknowledgements

Most of the text in this draft is from earlier contributions by Thomas Fossati and Klaus Hartke. The re-mix in this document is based on the requirements in [I-D.ietf-ace-coap-est], based on discussions with Michael Richardson, Panos Kampanis and Peter van der Stok.

## Author's Address

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Email: cabo@tzi.org



Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: July 10, 2014

T. Fossati  
Alcatel-Lucent  
P. Giacomin  
Freelance  
S. Loreto  
Ericsson  
January 06, 2014

Publish Option for CoAP  
draft-fossati-core-publish-option-03

Abstract

This memo defines the Publish Option for the Constrained Application Protocol (CoAP). The Publish Option is used by a CoAP Endpoint to control the authority delegation of one of its resources to another Endpoint. All the phases of the authority delegation process (setup, renewal, cancellation) are controlled by a simple RESTful protocol.

This memo also introduces the 'proxies' Web Linking relation type, to be used by a CoAP Proxy to explicitly advertise the resources that it can serve - either from its cache, or by forwarding the Client's request upstream.

The Publish Option and the 'proxies' relation provide the building blocks for a comprehensive, in-protocol, solution to the sleepy/intermittent Endpoint use case.

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 July 10, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .  | 3  |
| 1.1. Requirements Language and Motivation . . . . .                    | 3  |
| 2. Publish Option . . . . .  | 3  |
| 2.1. Value Format . . . . .  | 4  |
| 2.2. Operations . . . . .  | 4  |
| 2.2.1. Publishing a Resource . . . . .                                 | 4  |
| 2.2.2. Updating a Resource . . . . .                                   | 6  |
| 2.2.3. Unpublishing a Resource . . . . .                               | 6  |
| 2.2.4. Checking for Change . . . . .                                   | 7  |
| 3. The 'proxies' Relation Type . . . . .                               | 7  |
| 3.1. Examples . . . . .  | 7  |
| 3.1.1. Discover the Proxy for a Resource . . . . .                     | 8  |
| 3.1.2. Discover all the Resources that an Endpoint 'proxies' . . . . . | 8  |
| 3.2. Publish Link-Format Attributes . . . . .                          | 9  |
| 3.2.1. Implicitly . . . . .  | 9  |
| 3.2.2. Explicitly . . . . .  | 9  |
| 4. Acknowledgements . . . . .  | 10 |
| 5. IANA Considerations . . . . .                                       | 10 |
| 6. Security Considerations . . . . .                                   | 10 |
| 6.1. Securing the Delegation . . . . .                                 | 10 |
| 7. References . . . . .  | 11 |
| 7.1. Normative References . . . . .                                    | 11 |
| 7.2. Informative References . . . . .                                  | 11 |
| Appendix A. A (fairly) Comprehensive Example . . . . .                 | 11 |
| A.1. Actors . . . . .  | 12 |
| A.2. Resources . . . . .   | 12 |
| A.3. Application Flow . . . . .  | 12 |
| A.3.1. Bootstrap . . . . .   | 12 |
| A.3.2. Configuration and Reconfiguration . . . . .                     | 13 |
| A.3.3. Updating Functional Output . . . . .                            | 14 |
| A.3.4. Retrieving Functional Output . . . . .                          | 14 |
| A.3.5. SEP Reboot . . . . .  | 14 |
| Authors' Addresses . . . . .   | 15 |

## 1. Introduction

This memo defines the Publish Option for the Constrained Application Protocol [I-D.ietf-core-coap]. The Publish Option is used by a sleepy Endpoint (SEP) to temporarily delegate the authority of one of its resources to another, always on, Endpoint. The delegated Endpoint is typically a Proxy, though it could be an Endpoint with no other special network role. The SEP is given a simple RESTful messaging protocol that enables the setup, renewal and cancellation of the authority transfer. The whole process is driven by the SEP, which may actually never need to listen or to keep any state.

This memo also introduces the 'proxies' Web Linking [RFC5988] relation type. This new relation, which complements the default 'hosts' relation defined in [RFC6690], can be used by a CoAP Proxy to explicitly advertise the resources that it can serve, either from cache or by forwarding the Client's request upstream.

The 'proxies' relation works in concert with the Publish Option to enable SEP discovery even while SEP is off-line.

### 1.1. Requirements Language and Motivation

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 terms Client, Proxy, Server, and Endpoint are to be interpreted as described in [I-D.ietf-core-coap].

This memo reuses the terminology introduced in [I-D.rahman-core-sleepy-problem-statement], and aims at meeting the objectives stated in its Section 4 via an entirely in-protocol solution.

## 2. Publish Option

The Publish Option enables a SEP to temporarily (i.e. for a specified "lease" time) delegate the authority of one of its hosted resources to another Endpoint.

| No. | C | U | N | R | Name    | Format | Length | Default |
|-----|---|---|---|---|---------|--------|--------|---------|
| 31  | x | x | x | - | Publish | uint   | 1      | (none)  |

The one-byte integer value carried by the Publish Option allows the publishing node to specify the set of CoAP methods that are allowed on the resource (see Section 2.1 for details).

The "lease" time of the Publish action is specified by an associated (implicit or explicit) Max-Age Option value.

## 2.1. Value Format

The Publish Option consists of a single byte having the following layout:

```

  0 1 2 3 4 5 6 7
+---+---+---+---+
|G P D 0 0 0 0|
+---+---+---+---+

```

Each of the higher 3 bits is a flag field indicating whether the associated CoAP method (respectively: GET, PUT and DELETE) is allowed on the published resource. The POST method has resource/application specific semantics and can't therefore be safely delegated. The lower 5 bits are reserved and MUST be set to 0.

The 0x00 value is used to explicitly revoke the delegation (see Section 2.2.3.) and MUST NOT be used for any other purpose of the Option.

If the delegated Proxy receives a request for the published resource with a method that is not compatible with the mask supplied by the SEP, it MUST respond with a 4.05 (Method Not Allowed) response code.

## 2.2. Operations

### 2.2.1. Publishing a Resource

The SEP publishes one of its hosted resources, specified by the enclosed Proxy-URI, by making a PUT to the Proxy with a Publish Option attached. The Publish Option value specifies the CoAP methods that Clients are allowed to use on the resource (see Section 2.1).

The example below shows a delegation where the GET and PUT methods are allowed, whereas DELETE is explicitly prohibited, meaning that a Client can only read and update the resource.

```

P          SEP
|          |
|  PUT    | Proxy-URI: coap://sep.example.org/res
|<-----+ Publish: 0xC0

```

|  |         |  |                            |
|--|---------|--|----------------------------|
|  | r       |  | Content-Format: text/plain |
|  |         |  | Max-Age: 1200              |
|  | 2.01    |  |                            |
|  | +-----> |  | ETag: 0xabcd               |
|  |         |  |                            |

The Proxy, which is voluntarily entrusted by the resource owner to act as the delegated origin for the "lease" time specified by Max-Age, replies with a 2.01 (Created) if the authority transfer succeeds. An exact duplicate of the submitted representation is created, and from now on it can be accessed via the delegated Proxy using the original URI encoded in a Proxy-Uri Option. If the Publish operation isn't successful (e.g. because the Proxy does not support Publish), then the origin transfer fails, and an appropriate response code is returned (e.g. 4.02 Bad Option).

If no Max-Age is given, a default of 3600 seconds MUST be assumed. The Max-Age value, either implicit or explicit, determines the lifetime of the origin delegation. When Max-Age is elapsed, the Proxy MUST delete the published resource value (and any associated link-format metadata) and fall back to its usual proxying function.

On successful delegation, the Proxy MUST generate a new ETag and return it in the 2.01 response to the Client; if the published resource can be UPDATE'd, then the Client SHOULD save the ETag value (see Section 2.2.4).

The returned ETag value represents the state of the resource at the time the Publish operation is performed. The Proxy MUST change its value whenever the underlying resource representation changes, e.g. if it gets UPDATE'd. The current ETag value SHOULD be included by the Proxy in all responses involving the published representation. The ETag can be used by SEP to make conditional requests to the Proxy to check whether the representation has changed (see Section 2.2.4 for details).

The Publish Option is critical, and MUST NOT be present in a response. If the Proxy does not recognize it, a 4.02 (Bad Option) MUST be returned to the Client. If the Option value is not correctly formatted (see Section 2.1), a 4.00 (Bad Request) MUST be returned to the Client. The Publish Option is not Safe-to-Forward, and neither is a Cache-Key.

Since the 2.01 is emitted, and for the duration of the delegation, any Client wishing to access the resource can do so by making a Proxy-URI request to the Proxy, which shall then serve the resource from its own storage.

An interesting outcome of this communication strategy is that the SEP may really never need to listen on its radio interface. However, ignoring the response status code from Proxy, as well as the ETag value in case of UPDATE-able resources, is not a safe practice and SHOULD not be used unless the consequences are fully understood.

Upon publishing, the Proxy MUST save the identity (e.g. the IP address) of the publishing SEP, and MUST use it to correctly authorise "maintenance" operations such as renewal or cancellation of the published resource. The SEP identity MUST be kept for the whole duration of the delegation (including any associated renewal) and can be forgotten as soon as the delegation vanishes, either implicitly or explicitly.

#### 2.2.2. Updating a Resource

In order to update the delegated resource state or to just extend the lease period, the SEP sends basically the same request (except for the possibly updated representation value) to the Proxy, which in turn replies with a 2.04 Changed status code, and a new ETag value, in case the update operation succeeds. If the operation fails, e.g. because the request comes from an Endpoint different from the publishing SEP, a suitable status code is returned (e.g. 4.01 Unauthorized).

|   |         |                                       |
|---|---------|---------------------------------------|
| P |         | SEP                                   |
|   | PUT     | Proxy-URI: coap://sep.example.org/res |
|   | <-----  | Publish: 0xC0                         |
|   | r       | Content-Format: text/plain            |
|   |         | Max-Age: 1200                         |
|   | 2.04    |                                       |
|   | +-----> | ETag: 0xdcba                          |
|   |         |                                       |

#### 2.2.3. Unpublishing a Resource

The delegation of a given resource can be explicitly revoked by the SEP at any time before the lease time expires, by issuing a DELETE request to the Proxy hosting the resource duplicate with a Publish Option with value 0x00.

On successful deletion of the delegation, a 2.02 Deleted response code is returned by the Proxy. On error a suitable status code is returned.

```

P      SEP
|  DELETE  | Proxy-URI: coap://sep.example.org/res
|<-----+ Publish: 0x00
|
|  2.02    |
|+----->|
|          |

```

#### 2.2.4. Checking for Change

In order to check whether an UPDATE-able resource has changed, SEP issues a GET for the published resource with If-Match Option set to the last seen ETag value.

The possible outcomes are:

- o 4.04 (Not Found) if the resource has been deleted;
- o 2.05 (Content) if it has been otherwise modified;
- o 2.03 (Valid) if it has not changed.

In case a 2.05 is returned, SEP saves the updated ETag returned by the Proxy, and uses it on subsequent If-Match GET's.

Note that, in exceptionally simple scenarios, an unconditional GET followed by a memcmp against the previous representation value, MAY constitute a viable alternative to the method described above.

### 3. The 'proxies' Relation Type

The new 'proxies' Web Linking [RFC5988] relation type is meant to signify that the target resource carried by the link, which MUST be identified by an absolute URI, is reachable through a Proxy-URI request made to the anchored Origin (i.e. the Proxy).

(Note that we need to specify the Proxy through an explicit anchor, thus increasing the verbosity of the link value, because of the way the context URI override rules are defined in Section 2.1 of [RFC6690]. In fact, absent an explicit anchor, rule (b) would set the context to the SEP origin, which is definitely not what we want.)

#### 3.1. Examples

## 3.1.1. Discover the Proxy for a Resource

C multicasts a query to the /.well-known/core interface and discovers the P (associated to the coap://proxy.example.org authority) "proxies" the resource queried via an explicit href:

```

M      GET      C
|-----+-----|
|      | Uri-Path: .well-known
|      | Uri-Path: core
|      | Uri-Query: href="coap://sep.example.org/res"
P 2.05
+-----+----->
|      | <coap://sep.example.org/res>;
|      |   anchor="coap://proxy.example.org/";
|      |   rel="proxies"

```

## 3.1.2. Discover all the Resources that an Endpoint 'proxies'

C discovers all the resources that P "proxies":

```

P      GET      C
|-----+-----|
|      | Uri-Path: .well-known
|      | Uri-Path: core
|      | Uri-Query: rel="proxies"
P 2.05
+-----+----->
|      | <coap://sep.example.org/res>;
|      |   anchor="coap://proxy.example.org/";
|      |   rel="proxies",
|      | <...

```

and can then GET one of the "proxied" resource from P:

```

P      GET      C
|-----+-----|
|      | Proxy-URI: coap://sep.example.org/res
P 2.05
+-----+----->
|      | "res" data...

```

The 'proxies' relation is orthogonal to the Publish Option, so it's up to P to decide whether to serve coap://sep.example.org/res from its store/cache, or to forward the request to the origin at coap://sep.example.org.



### 3.2. Publish Link-Format Attributes

#### 3.2.1. Implicitly

The resource metadata are implicitly extracted from the published representation. Basically, the Proxy works out the 'ct' and 'sz' attributes by inspecting Content-Format and the request payload size.

The main advantage of this method is that it needs no further transmission except that needed for the Publish operation. The disadvantage is the very limited (and fixed) number of attributes that can be derived, which makes it suitable only for the most basic use cases.

#### 3.2.2. Explicitly

The resource metadata are explicitly published to the same Proxy-URI used for the sibling resource, either in a separate request/response cycle:

|   |          |   |
|---|----------|---|
| P |          | S                                       |
|   | PUT      | Proxy-URI: coap://sep.example.org/res   |
|   | <-----+< | Publish: 0x60                           |
|   | <meta>   | Content-Format: application/link-format |
|   | 2.01     |   |
|   | +----->  |   |
|   |          |   |

or atomically, within the same Publish operation, e.g. by using the Multipart Content-Format to aggregate one (or even more than one) representation(s) together with the application/link-format entry:

|   |          |   |
|---|----------|---|
| P |          | S   |
|   | PUT      | Proxy-URI: coap://sep.example.org/res         |
|   | <-----+< | Publish: 0x60                                 |
|   | [mp]     | Content-Format: application/multipart+publish |
|   | 2.01     | Max-Age: 1200                                 |
|   | +----->  |   |
|   |          |   |

Note that the former is non-atomic, and limited to only one representation of the resource; the latter is atomic and supports multiple Content-Format's for the published resource.

#### 4. Acknowledgements

Thanks to Bruce Nordman, Matthieu Vial, Akbar Rahman, and Esko Dijk for comments and discussions that have helped shaping this document.

#### 5. IANA Considerations

The following entry is added to the CoAP Option Numbers registry:

| Number | Name    | Reference |
|--------|---------|-----------|
| 31     | Publish | This memo |

This memo registers the new "proxies" Web Linking relation type as per [RFC5988].

Relation Name: proxies

Description: the target is the absolute URI of a resource proxied by the Origin stated in the anchor.

Reference: this memo

Notes: This relation is used in CoRE where links are retrieved as a `"/.well-known/core"` resource representation.

Application Data: None

#### 6. Security Considerations

This section identifies Threats (T) and related countermeasures (C).

- o T: cache poisoning.
- o C: use strong auth to identify SEP.
- o T: unauthorized update or de-registration
- o C: strong auth to identify SEP.
- o T: Proxy resources' exhaustion.
- o C: use strong auth to identify SEP + quota limit.
- o T: Inject fake copies of the resource by a 3rd party.
- o C: use delegation scheme that bundles the identities of the SEP and the Proxy, together with the resource being delegated. A third party must be able to verify SEP and Proxy identities, maybe offline, and check the resource fingerprint.

##### 6.1. Securing the Delegation

[[The following is just a sketch which needs further elaboration]]  
SEP signs the identity of the delegated Proxy and a fingerprint of the resource (both data and meta), and bundles it up with the resource itself, maybe in a MultiPart envelope (TBD define signed Content-Format). Client verifies the resource is indeed from the SEP by checking the signature, and it has been served by the intended origin, within the validity frame of the delegation. There seems to be an issue with hierarchical caching: the resource can't be served from a downstream Proxy which is different from the one that was originally delegated unless each Proxy in the delivery chain wraps the received message with its own credentials?

## 7. References

### 7.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

### 7.2. Informative References

- [I-D.rahman-core-sleepy-problem-statement]  
Rahman, A., Fossati, T., Loreto, S., and M. Vial, "Sleepy Devices in CoAP - Problem Statement", draft-rahman-core-sleepy-problem-statement-01 (work in progress), October 2012.

## Appendix A. A (fairly) Comprehensive Example

The following section details the whole life-cycle of an hypothetical Sleepy/Intermittent node that uses Publish to exchange data (both reading and writing) with other agents in a CoAP network.

### A.1. Actors

- SEP Sleeping/Intermittent endpoint implementing two functions: F1, and F2. Each function exposes one configurable parameter, and provides one output.
- P Proxy with Publish support.
- W Controller application which can configure function parameters on SEP.
- R Consumer application which reads values from SEP.

### A.2. Resources

The following resources model the two functions (F1 and F2) implemented by SEP in terms of their input and output parameters:

coap://sep1/i1 Configurable parameter for F1.

coap://sep1/i2 Configurable parameter for F2.

coap://sep1/o1 Output of F1.

coap://sep1/o2 Output of F2.

If the number of configuration parameters is not trivially small, then it might be handy to create an aux resource which can be polled by the SEP to track the parameters that have been reconfigured:

coap://sep1/im Update parameter mask. Conceptually a n-bit mask (one bit per configurable parameter) used by W to mark the updated parameters, and by SEP to clear them once the corresponding configuration has been applied.

### A.3. Application Flow

#### A.3.1. Bootstrap

SEP publishes all the application resources to P.

Configurable parameter for F1:

SEP -> P : PUT Proxy-URI=coap://sep1/i1, Publish="G,U", Payload="1"  
P -> SEP : 2.01 (Created), ETag=0x01

Configurable parameter for F2:

```
SEP -> P : PUT Proxy-URI=coap://sep1/i2, Publish="G,U", Payload="2"
P -> SEP : 2.01 (Created), ETag=0x01
```

Output of F1:

```
SEP -> P : PUT Proxy-URI=coap://sep1/o1, Publish="G", Payload=""
P -> SEP : 2.01 (Created), ETag=0x01
```

Output of F2:

```
SEP -> P : PUT Proxy-URI=coap://sep1/o2, Publish="G", Payload=""
P -> SEP : 2.01 (Created), ETag=0x01
```

This assumes that SEP has pre-canned values "1" and "2" for its configurable parameters i1 and i2 respectively.

Optionally:

```
SEP -> P : PUT Proxy-URI=coap://sep1/im, Publish="G,U",
          Payload="update_mask_cleared"
P -> SEP : 2.01 (Created), ETag=0x01
```

#### A.3.2. Configuration and Reconfiguration

W sets a new value, e.g. 5, for i2:

```
W -> P : PUT Proxy-URI=coap://sep1/i2, Payload="5"
P -> W : 2.04 (Changed), ETag=0x02
```

P updates the value of i2 accordingly, and sets a new ETag on it, e.g. 0x02.

When SEP wakes up, it polls its configuration variables via a conditional GET that uses the ETags returned by P at publishing time. Since i1 has not changed, and is still associated with the original ETag, a 2.03 status code is returned:

```
SEP -> P : GET Proxy-URI=coap://sep1/i1, If-Match=0x01
P -> SEP : 2.03 (Valid)
```

Since i2 has changed, a 2.05 status code is returned and the payload carries the new value. Also, the new ETag associated with i2 is returned and is updated locally by the SEP:

```
SEP -> P : GET Proxy-URI=coap://sep1/i2, If-Match=0x01
P -> SEP : 2.05 (Content), ETag=0x02, Payload="5"
```

The SEP reconfigures its F1 based on the new configuration setting, and continues its operations.

#### A.3.3. Updating Functional Output

SEP wakes up and commits the newly computed values, e.g. 6 and 8, to P:

```
SEP -> P : PUT Proxy-URI=coap://sep1/o1, Publish="G", Payload="6"
P -> SEP : 2.04 (Changed), ETag=0x02
```

```
SEP -> P : PUT Proxy-URI=coap://sep1/o2, Publish="G", Payload="8"
P -> SEP : 2.04 (Changed), ETag=0x02
```

P sets the new values, assigns a new ETag, and gives it back to P together with a 2.04 status code.

#### A.3.4. Retrieving Functional Output

R needs to retrieve the latest values for the functions computed by SEP; thus, it asks P to retrieve the associated resources:

```
R -> P : GET Proxy-URI=coap://sep1/o1
P -> R : 2.05 (Content), ETag=0x02, Payload="6"
```

```
R -> P : GET Proxy-URI=coap://sep1/o2
P -> R : 2.05 (Content), ETag=0x02, Payload="8"
```

Note that the exchange above applies to the very first poll. Subsequent polls can be done conditionally on the "last-seen" ETag.

Also note that the above assumes SEP has been able to update its values at least once. R must be prepared to retrieve empty representations, if SEP has not yet updated their value since bootstrap.

#### A.3.5. SEP Reboot

The idempotence of all the involved methods guarantees a clean recovery in face of a reboot of the SEP. In fact, if at a given time SEP reboots and loose soft state, including the configuration parameters: SEP has to go again through the bootstrap phase in which the application resources are published:

```
SEP -> P : PUT Proxy-URI=coap://sep1/i1, Publish="G,U", Payload="1"
P -> SEP : 2.04 (Changed), ETag=0x03
```

```
SEP -> P : PUT Proxy-URI=coap://sep1/i2, Publish="G,U", Payload="2"
```

P -> SEP : 2.04 (Changed), ETag=0x03

SEP -> P : PUT Proxy-URI=coap://sep1/o1, Publish="G", Payload=""

P -> SEP : 2.04 (Changed), ETag=0x03

SEP -> P : PUT Proxy-URI=coap://sep1/o2, Publish="G", Payload=""

P -> SEP : 2.04 (Changed), ETag=0x03

The ETag's value (0x03) needs to be not recently used (not within the Max-Age period for the resource).

From now on everything can proceed as described in Appendix A.3.2, Appendix A.3.3, and Appendix A.3.4

#### Authors' Addresses

Thomas Fossati  
Alcatel-Lucent  
3 Ely Road  
Milton, Cambridge CB24 6DD  
UK

Email: [thomas.fossati@alcatel-lucent.com](mailto:thomas.fossati@alcatel-lucent.com)

Pierpaolo Giacomini  
Freelance

Email: [yrz@anche.no](mailto:yrz@anche.no)

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: [salvatore.loreto@ericsson.com](mailto:salvatore.loreto@ericsson.com)

CoRE  
Internet-Draft  
Intended status: Informational  
Expires: March 15, 2014

O. Garcia-Morchon  
S. Kumar  
Philips Research  
S. Keoh  
University of Glasgow  
R. Hummen  
RWTH Aachen  
R. Struik  
Struik Consultancy  
September 11, 2013

Security Considerations in the IP-based Internet of Things  
draft-garcia-core-security-06

Abstract

A direct interpretation of the Internet of Things concept refers to the usage of standard Internet protocols to allow for human-to-thing or thing-to-thing communication. Although the security needs are well-recognized, it is still not fully clear how existing IP-based security protocols can be applied to this new setting. This Internet-Draft first provides an overview of security architecture, its deployment model and general security needs in the context of the lifecycle of a thing. Then, it presents challenges and requirements for the successful roll-out of new applications and usage of standard IP-based security protocols when applied to get a functional Internet of Things.

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 12, 2013.

Copyright Notice



Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

|  |    |
|--|----|
| 1. Conventions and Terminology Used in this Document . . . . .             | 4  |
| 2. Introduction . . . . .  | 4  |
| 3. The Thing Lifecycle and Architectural Considerations . . . . .          | 5  |
| 3.1. Threat Analysis . . . . .   | 6  |
| 3.2. Security Aspects . . . . .  | 10 |
| 4. State of the Art . . . . .  | 13 |
| 4.1. IP-based Security Solutions . . . . .                                 | 13 |
| 4.2. Wireless Sensor Network Security and Beyond . . . . .                 | 15 |
| 5. Challenges for a Secure Internet of Things . . . . .                    | 16 |
| 5.1. Constraints and Heterogeneous Communication . . . . .                 | 16 |
| 5.1.1. Tight Resource Constraints . . . . .                                | 16 |
| 5.1.2. Denial-of-Service Resistance . . . . .                              | 18 |
| 5.1.3. Protocol Translation and End-to-End Security . . . . .              | 18 |
| 5.2. Bootstrapping of a Security Domain . . . . .                          | 20 |
| 5.2.1. Distributed vs. Centralized Architecture and<br>Operation . . . . . | 20 |
| 5.2.2. Bootstrapping a thing's identity and keying<br>materials . . . . .  | 21 |
| 5.2.3. Privacy-aware Identification . . . . .                              | 22 |
| 5.3. Operation . . . . .   | 23 |
| 5.3.1. End-to-End Security . . . . .                                       | 23 |
| 5.3.2. Group Membership and Security . . . . .                             | 23 |
| 5.3.3. Mobility and IP Network Dynamics . . . . .                          | 24 |
| 6. Security Suites for the IP-based Internet of Things . . . . .           | 25 |
| 6.1. Security Architecture . . . . .                                       | 29 |
| 6.2. Security Model . . . . .  | 30 |
| 6.3. Security Bootstrapping and Management . . . . .                       | 31 |
| 6.4. Network Security . . . . .  | 33 |
| 6.5. Application Security . . . . .  | 34 |
| 7. Next Steps towards a Flexible and Secure Internet of Things .           | 36 |
| 8. Security Considerations . . . . .                                       | 40 |
| 9. IANA Considerations . . . . .   | 40 |
| 10. Acknowledgements . . . . .   | 40 |
| 11. References . . . . .   | 40 |
| 11.1. Informative References . . . . .                                     | 40 |
| Authors' Addresses . . . . .   | 45 |

## 1. Conventions and Terminology Used in this Document

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 "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

## 2. Introduction

The Internet of Things (IoT) denotes the interconnection of highly heterogeneous networked entities and networks following a number of communication patterns such as: human-to-human (H2H), human-to-thing (H2T), thing-to-thing (T2T), or thing-to-things (T2Ts). The term IoT was first coined by the Auto-ID center [AUTO-ID] in 1999. Since then, the development of the underlying concepts has ever increased its pace. Nowadays, the IoT presents a strong focus of research with various initiatives working on the (re)design, application, and usage of standard Internet technology in the IoT.

The introduction of IPv6 and web services as fundamental building blocks for IoT applications [RFC6568] promises to bring a number of basic advantages including: (i) a homogeneous protocol ecosystem that allows simple integration with Internet hosts; (ii) simplified development of very different appliances; (iii) an unified interface for applications, removing the need for application-level proxies. Such features greatly simplify the deployment of the envisioned scenarios ranging from building automation to production environments to personal area networks, in which very different things such as a temperature sensor, a luminaire, or an RFID tag might interact with each other, with a human carrying a smart phone, or with backend services.

This Internet Draft presents an overview of the security aspects of the envisioned all-IP architecture as well as of the lifecycle of an IoT device, a thing, within this architecture. In particular, we review the most pressing aspects and functionalities that are required for a secure all-IP solution.

With this, this Internet-Draft pursues several goals. First, we aim at presenting a comprehensive view of the interactions and relationships between an IoT application and security. Second, we aim at describing challenges for a secure IoT in the specific context of the lifecycle of a resource-constrained device. The final goal of this draft is to discuss the next steps towards a secure IoT.

The rest of the Internet-Draft is organized as follows. Section 3 depicts the lifecycle of a thing and gives general definitions for

the main security aspects within the IoT domain. In Section 4, we review existing protocols and work done in the area of security for wireless sensor networks. Section 5 identifies general challenges and needs for an IoT security protocol design and discusses existing protocols and protocol proposals against the identified requirements. Section 6 proposes a number of illustrative security suites describing how different applications involve distinct security needs. Section 7 includes final remarks and conclusions.

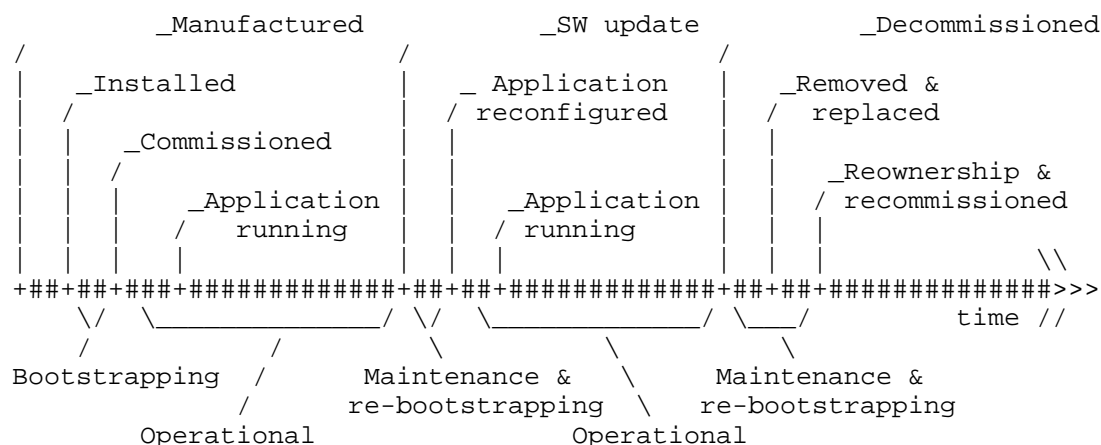
### 3. The Thing Lifecycle and Architectural Considerations

We consider the installation of a Building Automation and Control (BAC) system to illustrate the lifecycle of a thing in a BAC scenario. A BAC system consists of a network of interconnected nodes that perform various functions in the domains of HVAC (Heating, Ventilating, and Air Conditioning), lighting, safety etc. The nodes vary in functionality and a majority of them represent resource constrained devices such as sensors and luminaries. Some devices may also be battery operated or battery-less nodes, demanding for a focus on low energy consumption and on sleeping devices.

In our example, the life of a thing starts when it is manufactured. Due to the different application areas (i.e., HVAC, lighting, safety) nodes are tailored to a specific task. It is therefore unlikely that one single manufacturer will create all nodes in a building. Hence, interoperability as well as trust bootstrapping between nodes of different vendors is important. The thing is later installed and commissioned within a network by an installer during the bootstrapping phase. Specifically, the device identity and the secret keys used during normal operation are provided to the device during this phase. Different subcontractors may install different IoT devices for different purposes. Furthermore, the installation and bootstrapping procedures may not be a defined event but may stretch over an extended period of time. After being bootstrapped, the device and the system of things are in operational mode and run the functions of the BAC system. During this operational phase, the device is under the control of the system owner. For devices with lifetimes spanning several years, occasional maintenance cycles may be required. During each maintenance phase, the software on the device can be upgraded or applications running on the device can be reconfigured. The maintenance tasks can thereby be performed either locally or from a backend system. Depending on the operational changes of the device, it may be required to re-bootstrap at the end of a maintenance cycle. The device continues to loop through the operational phase and the eventual maintenance phase until the device is decommissioned at the end of its lifecycle. However, the end-of-life of a device does not necessarily mean that it is defective but

rather denotes a need to replace and upgrade the network to next-generation devices in order to provide additional functionality. Therefore the device can be removed and re-commissioned to be used in a different network under a different owner by starting the lifecycle over again. Figure 1 shows the generic lifecycle of a thing. This generic lifecycle is also applicable for IoT scenarios other than BAC systems.

At present, BAC systems use legacy building control standards such as BACNet [BACNET] or DALI [DALI] with independent networks for each subsystem (HVAC, lighting, etc.). However, this separation of functionality adds further complexity and costs to the configuration and maintenance of the different networks within the same building. As a result, more recent building control networks employ IP-based standards allowing seamless control over the various nodes with a single management system. While allowing for easier integration, this shift towards IP-based standards results in new requirements regarding the implementation of IP security protocols on constrained devices and the bootstrapping of security keys for devices across multiple manufacturers.



The lifecycle of a thing in the Internet of Things.

Figure 1

### 3.1. Threat Analysis

This section explores the security threats and vulnerabilities of a network of things in the IoTs. Security threats have been analyzed in related IP protocols including HTTPS [RFC2818], 6LoWPAN [RFC4919], ANCP [RFC5713], DNS security threats [RFC3833], SIP [RFC3261], IPv6

ND [RFC3756], and PANA [RFC4016]. Nonetheless, the challenge is about their impacts on scenarios of the IoTs. In this section, we specifically discuss the threats that could compromise an individual thing, or network as a whole, with regard to different phases in the thing's lifecycle. Note that these set of threats might go beyond the scope of Internet protocols but we gather them here for the sake of completeness.

- 1    Cloning of things: During the manufacturing process of a thing, an untrusted manufacturer can easily clone the physical characteristics, firmware/software, or security configuration of the thing. Subsequently, such a cloned thing may be sold at a cheaper price in the market, and yet be still able to function normally, as a genuine thing. For example, two cloned devices can still be associated and work with each other. In the worst case scenario, a cloned device can be used to control a genuine device. One should note here, that an untrusted manufacturer may also change functionality of the cloned thing, resulting in degraded functionality with respect to the genuine thing (thereby, inflicting potential reputational risk to the original thing manufacturer). Moreover, it can implement additional functionality with the cloned thing, such as a backdoor.
- 2    Malicious substitution of things: During the installation of a thing, a genuine thing may be substituted with a similar variant of lower quality without being detected. The main motivation may be cost savings, where the installation of lower-quality things (e.g., non-certified products) may significantly reduce the installation and operational costs. The installers can subsequently resell the genuine things in order to gain further financial benefits. Another motivation may be to inflict reputational damage on a competitor's offerings.
- 3    Eavesdropping attack: During the commissioning of a thing into a network, it may be susceptible to eavesdropping, especially if operational keying materials, security parameters, or configuration settings, are exchanged in clear using a wireless medium. After obtaining the keying material, the attacker might be able to recover the secret keys established between the communicating entities (e.g., H2T, T2Ts, or Thing to the backend management system), thereby compromising the authenticity and confidentiality of the communication channel, as well as the authenticity of commands and other traffic exchanged over this communication channel. When the network is in operation, T2T communication may be eavesdropped upon if the communication channel is not sufficiently protected or in the event of session key compromise due to a long period of usage without key renewal or updates.

- 4    Man-in-the-middle attack: The commissioning phase may also be vulnerable to man-in-the-middle attacks, e.g., when keying material between communicating entities is exchanged in the clear and the security of the key establishment protocol depends on the tacit assumption that no third party is able to eavesdrop on or sit in between the two communicating entities during the execution of this protocol. Additionally, device authentication or device authorization may be nontrivial, or may need support of a human decision process, since things usually do not have a priori knowledge about each other and can, therefore, not always be able to differentiate friends and foes via completely automated mechanisms. Thus, even if the key establishment protocol provides cryptographic device authentication, this knowledge on device identities may still need complementing with a human-assisted authorization step (thereby, presenting a weak link and offering the potential of man-in-the-middle attacks this way).
- 5    Firmware Replacement attack: When a thing is in operation or maintenance phase, its firmware or software may be updated to allow for new functionality or new features. An attacker may be able to exploit such a firmware upgrade by replacing the thing's with malicious software, thereby influencing the operational behaviour of the thing. For example, an attacker could add a piece of malicious code to the firmware that will cause it to periodically report the energy usage of the lamp to a data repository for analysis.
- 6    Extraction of security parameters: A thing deployed in the ambient environment (such as sensors, actuators, etc.) is usually physically unprotected and could easily be captured by an attacker. Such an attacker may then attempt to extract security information such as keys (e.g., device's key, private-key, group key) from this thing or try and re-program it to serve his needs. If a group key is used and compromised this way, the whole network may be compromised as well. Compromise of a thing's unique key has less security impact, since only the communication channels of this particular thing in question are compromised. Here, one should caution that compromise of the communication channel may also compromise all data communicated over this channel. In particular, one has to be weary of, e.g., compromise of group keys communicated over this channel (thus, leading to transitive exposure ripple effects).
- 7    Routing attack: As highlighted in [ID-Daniel], routing information in IoT can be spoofed, altered, or replayed, in order to create routing loops, attract/repel network traffic, extend/shorten source routes, etc. Other relevant routing attacks

include 1) Sinkhole attack (or blackhole attack), where an attacker declares himself to have a high-quality route/path to the base station, thus allowing him to do anything to all packets passing through it. 2) Selective forwarding, where an attacker may selectively forward packets or simply drop a packet. 3) Wormhole attack, where an attacker may record packets at one location in the network and tunnel them to another location, thereby influencing perceived network behaviour and potentially distorting statistics, thus greatly impacting the functionality of routing. 4) Sybil attack, whereby an attacker presents multiple identities to other things in the network.

- 8 Privacy threat: The tracking of a thing's location and usage may pose a privacy risk to its users. An attacker can infer information based on the information gathered about individual things, thus deducing behavioural patterns of the user of interest to him. Such information can subsequently be sold to interested parties for marketing purposes and targeted advertizing.
- 9 Denial-of-Service attack: Typically, things have tight memory and limited computation, they are thus vulnerable to resource exhaustion attack. Attackers can continuously send requests to be processed by specific things so as to deplete their resources. This is especially dangerous in the IoTs since an attacker might be located in the backend and target resource-constrained devices in an LLN. Additionally, DoS attack can be launched by physically jamming the communication channel, thus breaking down the T2T communication channel. Network availability can also be disrupted by flooding the network with a large number of packets.

The following table summarizes the security threats we identified above and the potential point of vulnerabilities at different layers of the communication stack. We also include related RFCs that include a threat model that might apply to the IoTs.



|                   | Manufacturing  | Installation/<br>Commissioning                 | Operation  |
|-------------------|----------------|--|--|
| Thing's Model     | Device Cloning | Substitution                                   | Privacy threat<br>Extraction of security params  |
| Application Layer |                | RFC2818<br>RFC4016                             | RFC2818, Firmware replacement                    |
| Transport Layer   |                | Eavesdropping &<br>Man-in-the-middle           | Eavesdropping<br>Man-in-the-middle               |
| Network Layer     |                | attack<br>RFC4919, RFC5713<br>RFC3833, RFC3756 | RFC4919, DoS attack<br>Routing attack<br>RFC3833 |
| Physical Layer    |                |  | DoS attack                                       |

The security threat analysis

Figure 2

### 3.2. Security Aspects

The term security subsumes a wide range of different concepts. In the first place, it refers to the basic provision of security services including confidentiality, authentication, integrity, authorization, non-repudiation, and availability, and some augmented services, such as duplicate detection and detection of stale packets (timeliness). These security services can be implemented by a combination of cryptographic mechanisms, such as block ciphers, hash functions, or signature algorithms, and non-cryptographic mechanisms, which implement authorization and other security policy enforcement aspects. For each of the cryptographic mechanisms, a solid key management infrastructure is fundamental to handling the required cryptographic keys, whereas for security policy enforcement, one needs to properly codify authorizations as a function of device roles and a security policy engine that implements these authorization checks and that can implement changes hereto throughout the system's lifecycle.

In the context of the IoT, however, the security must not only focus on the required security services, but also how these are realized in the overall system and how the security functionalities are executed.

To this end, we use the following terminology to analyze and classify security aspects in the IoT:

- 1    The security architecture refers to the system elements involved in the management of the security relationships between things and the way these security interactions are handled (e.g., centralized or distributed) during the lifecycle of a thing.
- 2    The security model of a node describes how the security parameters, processes, and applications are managed in a thing. This includes aspects such as process separation, secure storage of keying materials, etc.
- 3    Security bootstrapping denotes the process by which a thing securely joins the IoT at a given location and point in time. Bootstrapping includes the authentication and authorization of a device as well as the transfer of security parameters allowing for its trusted operation in a given network.
- 4    Network security describes the mechanisms applied within a network to ensure trusted operation of the IoT. Specifically, it prevents attackers from endangering or modifying the expected operation of networked things. Network security can include a number of mechanisms ranging from secure routing to data link layer and network layer security.
- 5    Application security guarantees that only trusted instances of an application running in the IoT can communicate with each other, while illegitimate instances cannot interfere.

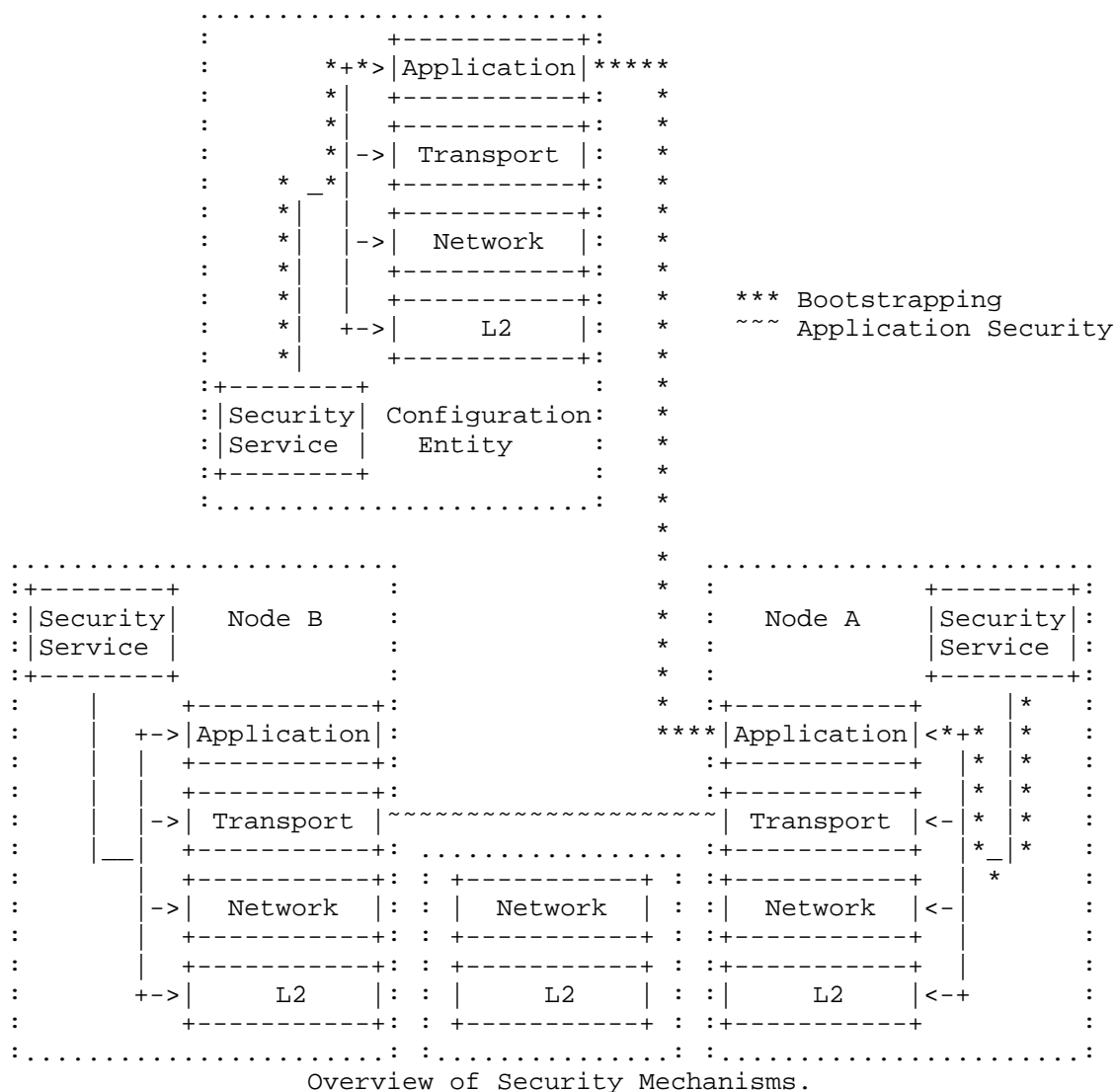


Figure 3

We now discuss an exemplary security architecture relying on a configuration entity for the management of the system with regard to the introduced security aspects (see Figure 2). Inspired by the security framework for routing over low power and lossy network [ID-Tsao], we show an example of security model and illustrates how different security concepts and the lifecycle phases map to the Internet communication stack. Assume a centralized architecture in

which a configuration entity stores and manages the identities of the things associated with the system along with their cryptographic keys. During the bootstrapping phase, each thing executes the bootstrapping protocol with the configuration entity, thus obtaining the required device identities and the keying material. The security service on a thing in turn stores the received keying material for the network layer and application security mechanisms for secure communication. Things can then securely communicate with each other during their operational phase by means of the employed network and application security mechanisms.

#### 4. State of the Art

Nowadays, there exists a multitude of control protocols for the IoT. For BAC systems, the ZigBee standard [ZB], BACNet [BACNET], or DALI [DALI] play key roles. Recent trends, however, focus on an all-IP approach for system control.

In this setting, a number of IETF working groups are designing new protocols for resource constrained networks of smart things. The 6LoWPAN working group [WG-6LoWPAN] concentrates on the definition of methods and protocols for the efficient transmission and adaptation of IPv6 packets over IEEE 802.15.4 networks [RFC4944]. The CoRE working group [WG-CoRE] provides a framework for resource-oriented applications intended to run on constrained IP network (6LoWPAN). One of its main tasks is the definition of a lightweight version of the HTTP protocol, the Constrained Application Protocol (CoAP) [ID-CoAP], that runs over UDP and enables efficient application-level communication for things.

##### 4.1. IP-based Security Solutions

In the context of the IP-based IoT solutions, consideration of TCP/IP security protocols is important as these protocols are designed to fit the IP network ideology and technology. While a wide range of specialized as well as general-purpose key exchange and security solutions exist for the Internet domain, we discuss a number of protocols and procedures that have been recently discussed in the context of the above working groups. The considered protocols are IKEv2/IPsec [RFC4306], TLS/SSL [RFC5246], DTLS [RFC5238], HIP [RFC5201][ID-Moskowitz], PANA [RFC5191], and EAP [RFC3748] in this Internet-Draft. Application layer solutions such as SSH [RFC4251] also exist, however, these are currently not considered. Figure 3 depicts the relationships between the discussed protocols in the context of the security terminology introduced in Section 3.1.

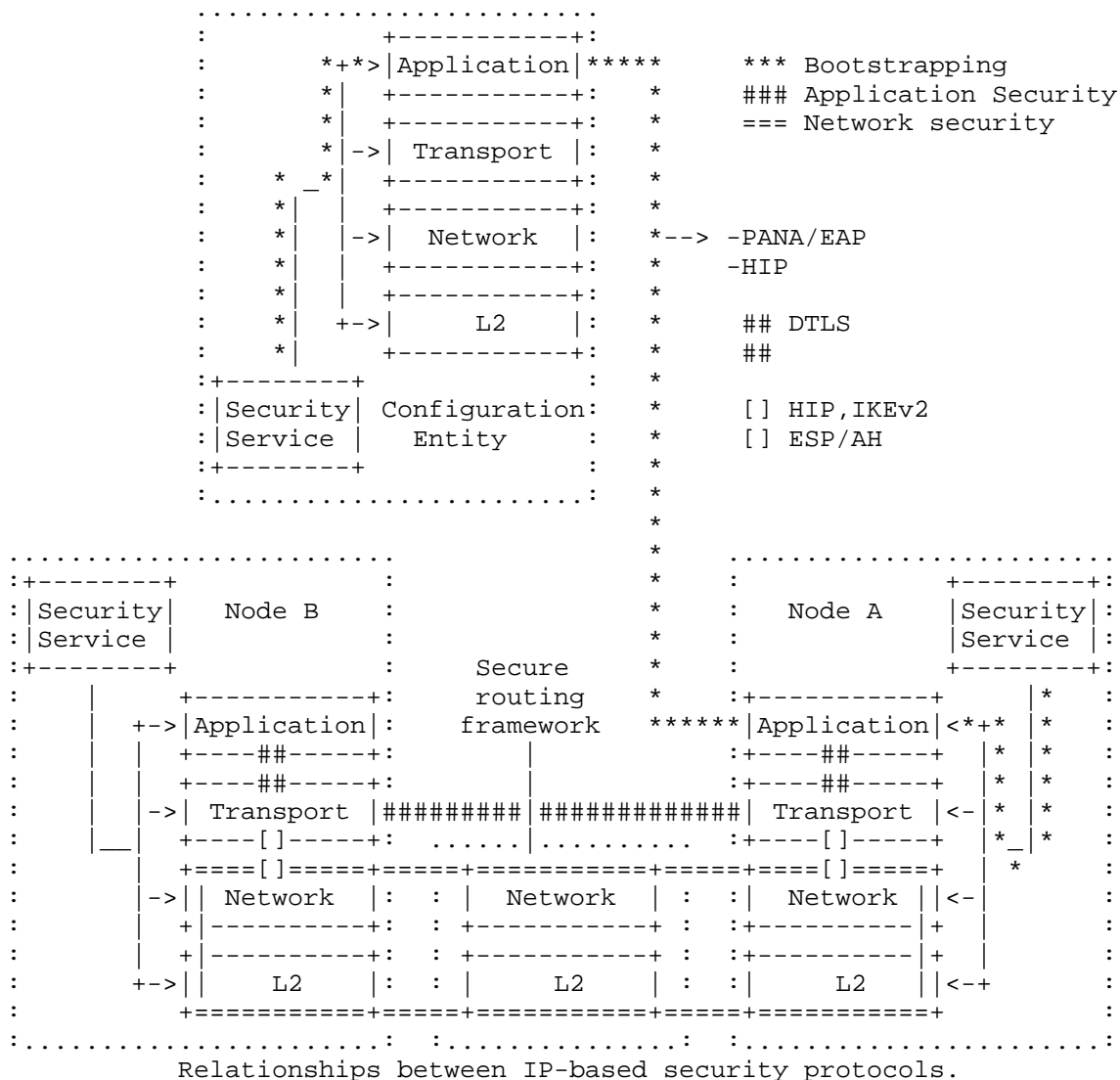


Figure 4

The Internet Key Exchange (IKEv2)/IPsec and the Host Identity protocol (HIP) reside at or above the network layer in the OSI model. Both protocols are able to perform an authenticated key exchange and set up the IPsec transforms for secure payload delivery. Currently, there are also ongoing efforts to create a HIP variant coined Diet HIP [ID-HIP] that takes lossy low-power networks into account at the authentication and key exchange level.

Transport Layer Security (TLS) and its datagram-oriented variant DTLS secure transport-layer connections. TLS provides security for TCP and requires a reliable transport, while DTLS secures and uses datagram-oriented protocols such as UDP. Both protocols are intentionally kept similar and share the same ideology and cipher suites.

The Extensible Authentication Protocol (EAP) is an authentication framework supporting multiple authentication methods. EAP runs directly over the data link layer and, thus, does not require the deployment of IP. It supports duplicate detection and retransmission, but does not allow for packet fragmentation. The Protocol for Carrying Authentication for Network Access (PANA) is a network-layer transport for EAP that enables network access authentication between clients and the network infrastructure. In EAP terms, PANA is a UDP-based EAP lower layer that runs between the EAP peer and the EAP authenticator.

#### 4.2. Wireless Sensor Network Security and Beyond

A variety of key agreement and privacy protection protocols that are tailored to IoT scenarios have been introduced in the literature. For instance, random key pre-distribution schemes [PROC-Chan] or more centralized solutions, such as SPINS [JOURNAL-Perrig], have been proposed for key establishment in wireless sensor networks. The ZigBee standard [ZB] for sensor networks defines a security architecture based on an online trust center that is in charge of handling the security relationships within a ZigBee network. Personal privacy in ubiquitous computing has been studied extensively, e.g., in [THESIS-Langheinrich]. Due to resource constraints and the specialization to meet specific requirements, these solutions often implement a collapsed cross layer optimized communication stack (e.g., without task-specific network layers and layered packet headers). Consequently, they cannot directly be adapted to the requirements of the Internet due to the nature of their design.

Despite important steps done by, e.g., Gupta et al. [PROC-Gupta], to show the feasibility of an end-to-end standard security architecture for the embedded Internet, the Internet and the IoT domain still do not fit together easily. This is mainly due to the fact that IoT security solutions are often tailored to the specific scenario requirements without considering interoperability with Internet protocols. On the other hand, the direct use of existing Internet security protocols in the IoT might lead to inefficient or insecure operation as we show in our discussion below.

## 5. Challenges for a Secure Internet of Things

In this section, we take a closer look at the various security challenges in the operational and technical features of the IoT and then discuss how existing Internet security protocols cope with these technical and conceptual challenges through the lifecycle of a thing. Table 1 summarizes which requirements need to be met in the lifecycle phases as well as the considered protocols. The structure of this section follows the structure of the table. This discussion should neither be understood as a comprehensive evaluation of all protocols, nor can it cover all possible aspects of IoT security. Yet, it aims at showing concrete limitations of existing Internet security protocols in some areas rather than giving an abstract discussion about general properties of the protocols. In this regard, the discussion handles issues that are most important from the authors' perspectives.

### 5.1. Constraints and Heterogeneous Communication

Coupling resource constrained networks and the powerful Internet is a challenge because the resulting heterogeneity of both networks complicates protocol design and system operation. In the following we briefly discuss the resource constraints of IoT devices and the consequences for the use of Internet Protocols in the IoT domain.

#### 5.1.1. Tight Resource Constraints

The IoT is a resource-constrained network that relies on lossy and low-bandwidth channels for communication between small nodes, regarding CPU, memory, and energy budget. These characteristics directly impact the threats to and the design of security protocols for the IoT domain. First, the use of small packets, e.g., IEEE 802.15.4 supports 127-byte sized packets at the physical layer, may result in fragmentation of larger packets of security protocols. This may open new attack vectors for state exhaustion DoS attacks, which is especially tragic, e.g., if the fragmentation is caused by large key exchange messages of security protocols. Moreover, packet fragmentation commonly downgrades the overall system performance due to fragment losses and the need for retransmissions. For instance, fate-sharing packet flight as implemented by DTLS might aggravate the resulting performance loss.

|              | Bootstrapping phase   | Operational Phase  |
|--------------|---|--|
| Requirements | Incremental deployment<br>Identity and key management<br>Privacy-aware identification<br>Group creation | End-to-End security<br>Mobility support<br>Group membership management |
| Protocols    | IKEv2<br>TLS/DTLS<br>HIP/Diet-HIP<br>PANA/EAP   | IKEv2/MOBIKE<br>TLS/DTLS<br>HIP/Diet-HIP                               |

Relationships between IP-based security protocols.

Figure 5

The size and number of messages should be minimized to reduce memory requirements and optimize bandwidth usage. In this context, layered approaches involving a number of protocols might lead to worse performance in resource-constrained devices since they combine the headers of the different protocols. In some settings, protocol negotiation can increase the number of exchanged messages. To improve performance during basic procedures such as, e.g., bootstrapping, it might be a good strategy to perform those procedures at a lower layer.

Small CPUs and scarce memory limit the usage of resource-expensive cryptoprimitives such as public-key cryptography as used in most Internet security standards. This is especially true, if the basic cryptoblocks need to be frequently used or the underlying application demands a low delay.

Independently from the development in the IoT domain, all discussed security protocols show efforts to reduce the cryptographic cost of the required public-key-based key exchanges and signatures with ECC[RFC5246][RFC5903][ID-Moskowitz][ID-HIP]. Moreover, all protocols have been revised in the last years to enable crypto agility, making cryptographic primitives interchangeable. Diet HIP takes the reduction of the cryptographic load one step further by focusing on cryptographic primitives that are to be expected to be enabled in hardware on IEEE 802.15.4 compliant devices. For example, Diet HIP does not require cryptographic hash functions but uses a CMAC [NIST] based mechanism, which can directly use the AES hardware available in standard sensor platforms. However, these improvements are only a first step in reducing the computation and communication overhead of Internet protocols. The question remains if other approaches can be



applied to leverage key agreement in these heavily resource-constrained environments.

A further fundamental need refers to the limited energy budget available to IoT nodes. Careful protocol (re)design and usage is required to reduce not only the energy consumption during normal operation, but also under DoS attacks. Since the energy consumption of IoT devices differs from other device classes, judgments on the energy consumption of a particular protocol cannot be made without tailor-made IoT implementations.

#### 5.1.2. Denial-of-Service Resistance

The tight memory and processing constraints of things naturally alleviate resource exhaustion attacks. Especially in unattended T2T communication, such attacks are difficult to notice before the service becomes unavailable (e.g., because of battery or memory exhaustion). As a DoS countermeasure, DTLS, IKEv2, HIP, and Diet HIP implement return routability checks based on a cookie mechanism to delay the establishment of state at the responding host until the address of the initiating host is verified. The effectiveness of these defenses strongly depends on the routing topology of the network. Return routability checks are particularly effective if hosts cannot receive packets addressed to other hosts and if IP addresses present meaningful information as is the case in today's Internet. However, they are less effective in broadcast media or when attackers can influence the routing and addressing of hosts (e.g., if hosts contribute to the routing infrastructure in ad-hoc networks and meshes).

In addition, HIP implements a puzzle mechanism that can force the initiator of a connection (and potential attacker) to solve cryptographic puzzles with variable difficulties. Puzzle-based defense mechanisms are less dependent on the network topology but perform poorly if CPU resources in the network are heterogeneous (e.g., if a powerful Internet host attacks a thing). Increasing the puzzle difficulty under attack conditions can easily lead to situations, where a powerful attacker can still solve the puzzle while weak IoT clients cannot and are excluded from communicating with the victim. Still, puzzle-based approaches are a viable option for sheltering IoT devices against unintended overload caused by misconfigured or malfunctioning things.

#### 5.1.3. Protocol Translation and End-to-End Security

Even though 6LoWPAN and CoAP progress towards reducing the gap between Internet protocols and the IoT, they do not target protocol specifications that are identical to their Internet pendants due to

performance reasons. Hence, more or less subtle differences between IoT protocols and Internet protocols will remain. While these differences can easily be bridged with protocol translators at gateways, they become major obstacles if end-to-end security measures between IoT devices and Internet hosts are used.

Cryptographic payload processing applies message authentication codes or encryption to packets. These protection methods render the protected parts of the packets immutable as rewriting is either not possible because a) the relevant information is encrypted and inaccessible to the gateway or b) rewriting integrity-protected parts of the packet would invalidate the end-to-end integrity protection.

There are essentially four solutions for this problem:

- 1    Sharing symmetric keys with gateways enables gateways to transform (e.g., de-compress, convert, etc.) packets and re-apply the security measures after transformation. This method abandons end-to-end security and is only applicable to simple scenarios with a rudimentary security model.
- 2    Reusing the Internet wire format in the IoT makes conversion between IoT and Internet protocols unnecessary. However, it leads to poor performance because IoT specific optimizations (e.g., stateful or stateless compression) are not possible.
- 3    Selectively protecting vital and immutable packet parts with a MAC or with encryption requires a careful balance between performance and security. Otherwise, this approach will either result in poor performance (protect as much as possible) or poor security (compress and transform as much as possible).
- 4    Message authentication codes that sustain transformation can be realized by considering the order of transformation and protection (e.g., by creating a signature before compression so that the gateway can decompress the packet without recalculating the signature). This enables IoT specific optimizations but is more complex and may require application-specific transformations before security is applied. Moreover, it cannot be used with encrypted data because the lack of cleartext prevents gateways from transforming packets.

To the best of our knowledge, none of the mentioned security protocols provides a fully customizable solution in this problem space. In fact, they usually offer an end-to-end secured connection. An exception is the usage layered approach as might be PANA and EAP. In such a case, this configuration (i) allows for a number of configurations regarding the location of, e.g., the EAP authenticator

and authentication server and (ii) the layered architecture might allow for authentication at different places. The drawback of this approach, however, lies in its high signaling traffic volume compared to other approaches. Hence, future work is required to ensure security, performance and interoperability between IoT and the Internet.

## 5.2. Bootstrapping of a Security Domain

Creating a security domain from a set of previously unassociated IoT devices is a key operation in the lifecycle of a thing and in the IoT network. In this section, we discuss general forms of network operation, how to communicate a thing's identity and the privacy implications arising from the communication of this identity.

### 5.2.1. Distributed vs. Centralized Architecture and Operation

Most things might be required to support both centralized and distributed operation patterns. Distributed thing-to-thing communication might happen on demand, for instance, when two things form an ad-hoc security domain to cooperatively fulfill a certain task. Likewise, nodes may communicate with a backend service located in the Internet without a central security manager. The same nodes may also be part of a centralized architecture with a dedicated node being responsible for the security management for group communication between things in the IoT domain. In today's IoT, most common architectures are fully centralized in the sense that all the security relationships within a segment are handled by a central party. In the ZigBee standard, this entity is the trust center. Current proposals for 6LoWPAN/CoRE identify the 6LoWPAN Border Router (6LBR) as such a device.

A centralized architecture allows for central management of devices and keying materials as well as for the backup of cryptographic keys. However, it also imposes some limitations. First, it represents a single point of failure. This is a major drawback, e.g., when key agreement between two devices requires online connectivity to the central node. Second, it limits the possibility to create ad-hoc security domains without dedicated security infrastructure. Third, it codifies a more static world view, where device roles are cast in stone, rather than a more dynamic world view that recognizes that networks and devices, and their roles and ownership, may change over time (e.g., due to device replacement and hand-over of control).

Decentralized architectures, on the other hand, allow creating ad-hoc security domains that might not require a single online management entity and are operative in a much more stand-alone manner. The ad-hoc security domains can be added to a centralized architecture at a

later point in time, allowing for central or remote management.

#### 5.2.2. Bootstrapping a thing's identity and keying materials

Bootstrapping refers to the process by which a device is associated to another one, to a network, or to a system. The way it is performed depends upon the architecture: centralized or distributed. It is important to realize that bootstrapping may involve different types of information, ranging from network parameters and information on device capabilities and their presumed functionality, to management information related to, e.g., resource scheduling and trust initialization/management. Furthermore, bootstrapping may occur in stages during the lifecycle of a device and may include provisioning steps already conducted during device manufacturing (e.g., imprinting a unique identifier or a root certificate into a device during chip testing), further steps during module manufacturing (e.g., setting of application-based configurations, such as temperature read-out frequencies and push-thresholds), during personalization (e.g., fine-tuned settings depending on installation context), during hand-over (e.g., transfer of ownership from supplier to user), and, e.g., in preparation of operation in a specific network. In what follows, we focus on bootstrapping of security-related information, since bootstrapping of all other information can be conducted as ordinary secured communications, once a secure and authentic channel between devices has been put in place.

In a distributed approach, a Diffie-Hellman type of handshake can allow two peers to agree on a common secret. In general, IKEv2, HIP, TLS, DTLS, can perform key exchanges and the setup of security associations without online connections to a trust center. If we do not consider the resource limitations of things, certificates and certificate chains can be employed to securely communicate capabilities in such a decentralized scenario. HIP and Diet HIP do not directly use certificates for identifying a host, however certificate handling capabilities exist for HIP and the same protocol logic could be used for Diet HIP. It is noteworthy, that Diet HIP does not require a host to implement cryptographic hashes. Hence, some lightweight implementations of Diet HIP might not be able to verify certificates unless a hash function is implemented by the host.

In a centralized architecture, preconfigured keys or certificates held by a thing can be used for the distribution of operational keys in a given security domain. A current proposal [ID-OFlynn] refers to the use of PANA for the transport of EAP messages between the PANA client (the joining thing) and the PANA Authentication Agent (PAA), the 6LBR. EAP is thereby used to authenticate the identity of the joining thing. After the successful authentication, the PANA PAA

provides the joining thing with fresh network and security parameters.

IKEv2, HIP, TLS, and DTLS could be applied as well for the transfer of configuration parameters in a centralized scenario. While HIP's cryptographic secret identifies the thing, the other protocols do not represent primary identifiers but are used instead to bind other identifiers such as the operation keys to the public-key identities.

In addition to the protocols, operational aspects during bootstrapping are of key importance as well. Many other standard Internet protocols assume that the identity of a host is either available by using secondary services like certificate authorities or secure name resolution (e.g., DNSsec) or can be provided over a side channel (entering passwords via screen and keyboard). While these assumptions may hold in traditional networks, intermittent connectivity, localized communication, and lack of input methods complicate the situation for the IoT.

The order in which the things within a security domain are bootstrapped plays an important role as well. In [RFC6345], the PANA relay element is introduced, relaying PANA messages between a PaC (joining thing) and PAA of a segment [ID-OFlynn]. This approach forces commissioning based on distance to PAA, i.e., things can only be bootstrapped hop-by-hop starting from those closer to the PAA, all things that are 1-hop away are bootstrapped first, followed by those that are 2-hop away, and so on. Such an approach might impose important limitations on actual use cases in which, e.g., an installer without technical background has to roll-out the system.

#### 5.2.3. Privacy-aware Identification

During the last years, the introduction of RFID tags has raised privacy concerns because anyone might access and track tags. As the IoT involves not only passive devices, but also includes active and sensing devices, the IoT might irrupt even deeper in people's privacy spheres. Thus, IoT protocols should be designed to avoid these privacy threats during bootstrapping and operation where deemed necessary. In H2T and T2T interactions, privacy-aware identifiers might be used to prevent unauthorized user tracking. Similarly, authentication can be used to prove membership of a group without revealing unnecessary individual information.

TLS and DTLS provide the option of only authenticating the responding host. This way, the initiating host can stay anonymous. If authentication for the initiating host is required as well, either public-key certificates or authentication via the established encrypted payload channel can be employed. Such a setup allows to

only reveal the responder's identity to possible eavesdroppers.

HIP and IKEv2 use public-key identities to authenticate the initiator of a connection. These identities could easily be traced if no additional protection were in place. IKEv2 transmits this information in an encrypted packet. Likewise, HIP provides the option to keep the identity of the initiator secret from eavesdroppers by encrypting it with the symmetric key generated during the handshake. However, Diet HIP cannot provide a similar feature because the identity of the initiator simultaneously serves as static Diffie-Hellman key. Note that all discussed solutions could use anonymous public-key identities that change for each communication. However, such identity cycling may require a considerable computational effort for generating new asymmetric key pairs. In addition to the built-in privacy features of the here discussed protocols, a large body of anonymity research for key exchange protocols exists. However, the comparison of these protocols and protocol extensions is out of scope for this work.

### 5.3. Operation

After the bootstrapping phase, the system enters the operational phase. During the operational phase, things can relate to the state information created during the bootstrapping phase in order to exchange information securely and in an authenticated fashion. In this section, we discuss aspects of communication patterns and network dynamics during this phase.

#### 5.3.1. End-to-End Security

Providing end-to-end security is of great importance to address and secure individual T2T or H2T communication within one IoT domain. Moreover, end-to-end security associations are an important measure to bridge the gap between the IoT and the Internet. IKEv2 and HIP, TLS and DTLS provide end-to-end security services including peer entity authentication, end-to-end encryption and integrity protection above the network layer and the transport layer respectively. Once bootstrapped, these functions can be carried out without online connections to third parties, making the protocols applicable for decentralized use in the IoT. However, protocol translation by intermediary nodes may invalidate end-to-end protection measures (see Section 5.1).

#### 5.3.2. Group Membership and Security

In addition to end-to-end security, group key negotiation is an important security service for the T2Ts and Ts2T communication patterns in the IoT as efficient local broadcast and multicast relies

on symmetric group keys.

All discussed protocols only cover unicast communication and therefore do not focus on group-key establishment. However, the Diffie-Hellman keys that are used in IKEv2 and HIP could be used for group Diffie-Hellman key-negotiations. Conceptually, solutions that provide secure group communication at the network layer (IPsec/IKEv2, HIP/Diet HIP) may have an advantage regarding the cryptographic overhead compared to application-focused security solutions (TLS/DTLS). This is due to the fact that application-focused solutions require cryptographic operations per group application, whereas network layer approaches may allow to share secure group associations between multiple applications (e.g., for neighbor discovery and routing or service discovery). Hence, implementing shared features lower in the communication stack can avoid redundant security measures.

A number of group key solutions have been developed in the context of the IETF working group MSEC in the context of the MIKEY architecture [WG-MSEC][RFC4738]. These are specifically tailored for multicast and group broadcast applications in the Internet and should also be considered as candidate solutions for group key agreement in the IoT. The MIKEY architecture describes a coordinator entity that disseminates symmetric keys over pair-wise end-to-end secured channels. However, such a centralized approach may not be applicable in a distributed environment, where the choice of one or several coordinators and the management of the group key is not trivial.

#### 5.3.3. Mobility and IP Network Dynamics

It is expected that many things (e.g., wearable sensors, and user devices) will be mobile in the sense that they are attached to different networks during the lifetime of a security association. Built-in mobility signaling can greatly reduce the overhead of the cryptographic protocols because unnecessary and costly re-establishments of the session (possibly including handshake and key agreement) can be avoided. IKEv2 supports host mobility with the MOBIKE [RFC4555][RFC4621] extension. MOBIKE refrains from applying heavyweight cryptographic extensions for mobility. However, MOBIKE mandates the use of IPsec tunnel mode which requires to transmit an additional IP header in each packet. This additional overhead could be alleviated by using header compression methods or the Bound End-to-End Tunnel (BEET) mode [ID-Nikander], a hybrid of tunnel and transport mode with smaller packet headers.

HIP offers a simple yet effective mobility management by allowing hosts to signal changes to their associations [RFC5206]. However, slight adjustments might be necessary to reduce the cryptographic

costs, for example, by making the public-key signatures in the mobility messages optional. Diet HIP does not define mobility yet but it is sufficiently similar to HIP to employ the same mechanisms. TLS and DTLS do not have standards for mobility support, however, work on DTLS mobility exists in the form of an Internet draft [ID-Williams]. The specific need for IP-layer mobility mainly depends on the scenario in which nodes operate. In many cases, mobility support by means of a mobile gateway may suffice to enable mobile IoT networks, such as body sensor networks. However, if individual things change their point of network attachment while communicating, mobility support may gain importance.

## 6. Security Suites for the IP-based Internet of Things

Different applications have different security requirements and needs and, depending on various factors, such as device capability, availability of network infrastructure, security services needed, usage, etc., the required security protection may vary from "no security" to "full-blown security". For example, applications may have different needs regarding authentication and confidentiality. While some application might not require any authentication at all, others might require strong end-to-end authentication. In terms of secure bootstrapping of keys, some applications might assume the existence and online availability of a central key-distribution-center (KDC) within the 6LoWPAN network to distribute and manage keys; while other applications cannot rely on such a central party or their availability.

Thus, it is essential to define security profiles to better tailor security solutions for different applications with the same characteristics and requirements. This provides a means of grouping applications into profiles and then defines the minimal required security primitives to enable and support the security needs of the profile. The security elements in a security profile can be classified according to Section 3.1, namely:

- 1    Security architecture,
- 2    Security model,
- 3    Security bootstrapping,
- 4    Network security, and



5    Application security.

In order to (i) guide the design process by identifying open gaps; (ii) allow for later interoperability; and (iii) prevent possible security misconfigurations, this section defines a number of generic security profiles with different security needs. Each security profile is identified by:

- 1    a short description,
- 2    an exemplary application that might use/require such a security policy,
- 3    the security requirements for each of the above security aspects according to our classification in Section 3.1.

These security profiles can serve to guide the standardization process, since these explicitly describe the basic functionalities and protocols required to get different use cases up and running. It can allow for later interoperability since different manufacturers can describe the implemented security profile in their products. Finally, the security profiles can avoid possible security misconfigurations, since each security profile can be bound to a different application area so that it can be clearly defined which security protocols and approaches can be applied where and under which circumstances.

Note that each of these security profiles aim at summarizing the required security requirements for different applications and at providing a set of initial security features. In other words, these profiles reflect the need for different security configurations, depending on the threat and trust models of the underlying applications. In this sense, this section does not provide an overview of existing protocols as done in previous sections of the Internet Draft, but it rather explicitly describes what should be in place to ensure secure system operation. Observe also that this list of security profiles is not exhaustive and that it should be considered just as an example not related to existing legal regulations for any existing application. These security profiles are summarized in the table below:

|           | Application               | Description  |
|-----------|---------------------------|--|
| SecProf_0 | No security needs         | 6LoWPAN/CoAP is used without security  |
| SecProf_1 | Home usage                | Enables operation between home things without interaction with central device                              |
| SecProf_2 | Managed Home usage        | Enables operation between home things. Interaction with a central and local device is possible             |
| SecProf_3 | Industrial usage          | Enables operation between things. Relies on central (local or backend) device for security                 |
| SecProf_4 | Advanced Industrial usage | Enables ad-hoc operation between things and relies on central device or on a collection of control devices |

Security profiles and application areas.

Figure 6

The classification in the table considers different potential applications and situations in which their security needs change due to different operational features (network size, existence of a central device, connectivity to the Internet, importance of the exchanged information, etc) or threat model (what are the assets that an attacker looks for). As already pointed out, this set of scenarios is exemplary and they should be further discussed based on a broader consensus.

SecProf\_0 is meant for any application that does not require security. Examples include applications during system development, system testing, or some very basic applications in which security is not required at all.

The second security suite (SecProf\_1) is catered for environments in which 6LoWPAN/CoAP can be used to enable communication between things in an ad-hoc manner and the security requirements are minimal. An example, is a home application in which two devices should exchange information and no further connection with other devices (local or with a backend) is required. In this scenario, value of the exchanged information is low and that it usually happen in a confined room, thus, it is possible to have a short period of time during

which initial secrets can be exchanged in the clear. Due to this fact, there is no requirement to enable devices from different manufacturers to interoperate in a secure way (keys are just exchanged). The expected network size of applications using this profile is expected to be small such that the provision of network security, e.g., secure routing, is of low importance.

The next security suite (SecProf\_2) represents an evolution of SecProf\_1 in which, e.g., home devices, can be managed locally. A first possibility for the securing domain management refers to the creation of a centrally managed security domain without any connectivity to the Internet. The central device used for management can serve as, e.g., a key distribution center including policies for key update, storage, etc. The presence of a central device can help in the management of larger networks. Network security becomes more relevant in this scenario since the 6LoWPAN/CoAP network can be prone to Denial of Service attacks (e.g., flooding if L2 is not protected) or routing attacks.

SecProf\_3 considers that a central device is always required for network management. Example applications of this profile include building control and automation, sensor networks for industrial use, environmental monitoring, etc. As before, the network manager can be located in the 6LoWPAN/CoAP network and handle key management. In this case, the first association of devices to the network is required to be done in a secure way. In other words, the threat model requires measurements to protect against any vulnerable period of time. This step can involve the secure transmission of keying materials used for network security at different layers. The information exchanged in the network is considered to be valuable and it should be protected in the sense of pairwise links. Commands should be secured and broadcast should be secured with entity authentication [ID-CoAPMulticast]. Network should be protected from attacks. A further extension to this use case is to allow for remote management. A "backend manager" is in charge of managing SW or information exchanged or collected within the 6LoWPAN/CoAP network. This requires connection of devices to the Internet over a 6LBR involving a number of new threats that were not present before. A list of potential attacks include: resource-exhaustion attacks from the Internet; amplification attacks; trust issues related a HTTP-CoAP proxy [ID-proHTTPCoAP], etc. This use case requires protecting the communication from a device in the backend to a device in the 6LoWPAN/CoAP network, end-to-end. This use case also requires measures to provide the 6LBR with the capability of dropping fake requests coming from the Internet. This becomes especially challenging when the 6LBR is not trusted and access to the exchanged information is limited; and even more in the case of a HTTP-CoAP proxy since protocol translation is required. This use case should

take care of protecting information accessed from the backend due to privacy issues (e.g., information such as type of devices, location, usage, type and amount of exchanged information, or mobility patterns can be gathered at the backend threatening the privacy sphere of users) so that only required information is disclosed.

The last security suite (SecProf\_4) essentially represents interoperability of all the security profiles defined previously. It considers applications with some additional requirements regarding operation such as: (i) ad-hoc establishment of security relationships between things (potentially from different manufacturers) in non-secure environments or (ii) dynamic roaming of things between different 6LoWPAN/CoAP security domains. Such operational requirements pose additional security requirements, e.g., in addition to secure bootstrapping of a device within a 6LoWPAN/CoAP security domain and the secure transfer of network operational key, there is a need to enable inter-domains secure communication to facilitate data sharing.

The above description illustrates how different applications of 6LoWPAN/CoAP networks involve different security needs. In the following sections, we summarize the expected security features or capabilities for each the security profile with regards to "Security Architecture", "Security Model", "Security Bootstrapping", "Network Security", and "Application Security".

#### 6.1. Security Architecture

The choice of security architecture has many implications regarding key management, access control, or security scope. A distributed (or ad-hoc) architecture means that security relationships between things are setup on the fly between a number of objects and kept in a decentralized fashion. A locally centralized security architecture means that a central device, e.g., the 6LBR, handles the keys for all the devices in the security domain. Alternatively, a central security architecture could also refer to the fact that smart objects are managed from the backend. The security architecture for the different security profiles is classified as follows.

|           | Description                                    |
|-----------|--|
| SecProf_0 | -  |
| SecProf_1 | Distributed                                    |
| SecProf_2 | Distributed able to move centralized (local)   |
| SecProf_3 | Centralized (local &/or backend)               |
| SecProf_4 | Distributed & centralized (local &/or backend) |

Security architectures in different security profiles.

Figure 7

In "SecProf\_1", management mechanisms for the distributed assignment and management of keying materials is required. Since this is a very simple use case, access control to the security domain can be enabled by means of a common secret known to all devices. In the next security suite (SecProf\_2), a central device can assume key management responsibilities and handle the access to the network. The last two security suites (SecProf\_3 and SecProf\_4) further allow for the management of devices or some keying materials from the backend.

## 6.2. Security Model

While some applications might involve very resource-constrained things such as, e.g., a humidity, pollution sensor, other applications might target more powerful devices aimed at more exposed applications. Security parameters such as keying materials, certificates, etc must be protected in the thing, for example by means of tamper-resistant hardware. Keys may be shared across a thing's networking stack to provide authenticity and confidentiality in each networking layer. This would minimize the number of key establishment/agreement handshake and incurs less overhead for constrained thing. While more advance applications may require key separation at different networking layers, and possibly process separation and sandboxing to isolate one application from another. In this sense, this section reflects the fact that different applications require different sets of security mechanisms.

|           | Description  |
|-----------|--|
| SecProf_0 | -  |
| SecProf_1 | No tamper resistant<br>Sharing keys between layers   |
| SecProf_2 | No tamper resistant<br>Sharing keys between layers   |
| SecProf_3 | Tamper resistant<br>Key and process separation   |
| SecProf_4 | (no) Tamper resistant<br>Sharing keys between layers/Key and process separation<br>Sandbox |

Thing security models in different security profiles.

Figure 8

### 6.3. Security Bootstrapping and Management

Bootstrapping refers to the process by which a thing initiates its life within a security domain and includes the initialization of secure and/or authentic parameters bound to the thing and at least one other device in the network. Here, different mechanisms may be used to achieve confidentiality and/or authenticity of these parameters, depending on deployment scenario assumptions and the communication channel(s) used for passing these parameters. The simplest mechanism for initial set-up of secure and authentic parameters is via communication in the clear using a physical interface (USB, wire, chip contact, etc.). Here, one commonly assumes this communication channel is secure, since eavesdropping and/or manipulation of this interface would generally require access to the physical medium and, thereby, to one or both of the devices themselves. This mechanism was used with the so-called original "resurrecting duckling" model, as introduced in [PROC-Stajano]. This technique may also be used securely in wireless, rather than wired, set-ups, if the prospect of eavesdropping and/or manipulating this channel are dim (a so-called "location-limited" channel [PROC-Smetters-04, PROC-Smetters-02]). Examples hereof include the communication of secret keys in the clear using near field communication (NFC) - where the physical channel is purported to have very limited range (roughly 10cm), thereby thwarting eavesdropping by

far-away adversarial devices, and in-the-clear communication during a small time window (triggered by, e.g., a button-push) - where eavesdropping is presumed absent during this small time window. With the use of public-key based techniques, assumptions on the communication channel can be relaxed even further, since then the cryptographic technique itself provides for confidentiality of the channel set-up and the location-limited channel - or use of certificates - rules out man-in-the-middle attacks, thereby providing authenticity [PROC-Smetters-02]. The same result can be obtained using password-based public-key protocols [SPEKE], where authenticity depends on the (weak) password not being guessed during execution of the protocol. It should be noted that while most of these techniques realize a secure and authentic channel for passing parameters, these generally do not provide for explicit authorization. As an example, with use of certificate-based public-key based techniques, one may obtain hard evidence on whom one shares secret and/or authentic parameters with, but this does not answer the question as to whether one wishes to share this information at all with this specifically identified device (the latter usually involves a human-decision element). Thus, the bootstrapping mechanisms above should generally be complemented by mechanisms that regulate (security policies for) authorization. Furthermore, the type of bootstrapping is very related to the required type of security architecture. Distributed bootstrapping means that a pair of devices can setup a security relationship on the fly, without interaction with a central device elsewhere within the system. In many cases, it is handy to have a distributed bootstrapping protocol based on existing security protocols (e.g., DTLS in CoAP) required for other purposes: this reduces the amount of required software. A centralized bootstrapping protocol is one in which a central device manages the security relationships within a network. This can happen locally, e.g., handled by the 6LBR, or remotely, e.g., from a server connected via the Internet. The security bootstrapping for the different security profiles is as follows.

|           | Description   |
|-----------|---|
| SecProf_0 | -   |
| SecProf_1 | * Distributed, (e.g., Resurrecting duckling)<br>* First key distribution happens in the clear   |
| SecProf_2 | * Distributed, (e.g., Resurrecting duckling )<br>* Centralized (local), 6LBR acts as KDC<br>* First key distribution occurs in the clear, if the KDC is available, the KDC can manage network access  |
| SecProf_3 | * 6LBR acts as KDC. It handles node joining, provides them with keying material from L2 to application layers<br>* Bootstrapping occurs in a secure way - either in secure environment or the security mechanisms ensure that eavesdropping is not possible.<br>* KDC and backend can implement secure methods for network access |
| SecProf_4 | * As in SecProf_3.  |

Security bootstrapping methods in different security profiles

Figure 9

#### 6.4. Network Security

Network security refers to the mechanisms used to ensure the secure transport of 6LoWPAN frames. This involves a multitude of issues ranging from secure discovery, frame authentication, routing security, detection of replay, secure group communication, etc. Network security is important to thwart potential attacks such as denial-of-service (e.g., through message flooding) or routing attacks.

The Internet Draft [ID-Tsao] presents a very good overview of attacks and security needs classified according to the confidentiality, integrity, and availability needs. A potential limitation is that there exist no differentiation in security between different use cases and the framework is limited to L3. The security suites gathered in the present ID aim at solving this by allowing for a more flexible selection of security needs at L2 and L3.



|           | Description   |
|-----------|---|
| SecProf_0 | -   |
| SecProf_1 | <ul style="list-style-type: none"> <li>* Network key creating a home security domain at L2 ensuring authentication and freshness of exchanged data</li> <li>* Secure multicast does not ensure origin authentication</li> <li>* No need for secure routing at L3</li> </ul>   |
| SecProf_2 | <ul style="list-style-type: none"> <li>* Network key creating a home security domain at L2 ensuring authentication and freshness of exchanged data</li> <li>* Secure multicast does not ensure origin authentication</li> <li>* No need for secure routing at L3</li> </ul>   |
| SecProf_3 | <ul style="list-style-type: none"> <li>* Network key creating an industry security domain at L2 ensuring authentication and freshness of exchanged data</li> <li>* Secure routing needed (integrity &amp; availability) at L3 within 6LoWPAN/CoAP</li> <li>* Secure multicast requires origin authentication</li> </ul>   |
| SecProf_4 | <ul style="list-style-type: none"> <li>* Network key creating an industry security domain at L2 ensuring authentication and freshness of exchanged data</li> <li>* Inter-domain authentication/secure handoff</li> <li>* Secure routing needed at L3</li> <li>* Secure multicast requires origin authentication</li> <li>* 6LBR (HTTP-CoAP proxy) requires verification of forwarded messages and messages leaving or entering the 6LoWPAN/CoAP network.</li> </ul> |

Network security needs in different security profiles

Figure 10

### 6.5. Application Security

In the context of 6LoWPAN/CoAP networks, application security refers firstly to the configuration of DTLS used to protect the exchanged information. It further refers to the measures required in potential translation points (e.g., a (HTTP-CoAP) proxy) where information can be collected and the privacy sphere of users in a given security domain is endangered. Application security for the different security profiles is as follows.

|           | Description  |
|-----------|--|
| SecProf_0 | -  |
| SecProf_1 | -  |
| SecProf_2 | <ul style="list-style-type: none"> <li>* DTLS is used for end-to-end application security between management device and things and between things</li> <li>* DTLS ciphersuites configurable to provide confidentiality and/or authentication and/or freshness</li> <li>* Key transport and policies for generation of session keys are required</li> </ul>   |
| SecProf_3 | <ul style="list-style-type: none"> <li>* Requirements as in SecProf_2 and</li> <li>* DTLS is used for end-to-end application security between management device and things and between things</li> <li>* Communication between KDC and each thing secured by pairwise keys</li> <li>* Group keys for communication in a group distributed by KDC</li> <li>* Privacy protection should be provided in translation points</li> </ul> |
| SecProf_4 | <ul style="list-style-type: none"> <li>* Requirements as in SecProf_3 and</li> <li>* TLS or DTLS can be used to send commands from the backend to the 6LBR or things in a 6LoWPAN/CoAP network</li> <li>* End-to-end secure connectivity from backend required</li> <li>* Secure broadcast in a network from backend required</li> </ul>   |

Application security methods in different security profiles

Figure 11

The first two security profiles do not include any security at the application layer. The reason is that, in the first case, security is not provided and, in the second case, it seems reasonable to provide basic security at L2. In the third security profile (SecProf\_2), DTLS becomes the way of protecting messages at application layer between things and with the KDC running on a 6LBR. A key option refers to the capability of easily configuring DTLS to provide a subset of security services (e.g., some applications do not require confidentiality) to reduce the impact of security in the system operation of resource-constrained things. In addition to basic key management mechanisms running within the KDC, communication protocols for key transport or key update are required. These

protocols could be based on DTLS. The next security suite (SecProf\_3) requires pairwise keys for communication between things within the security domain. Furthermore, it can involve the usage of group keys for group communication. If secure multicast is implemented, it should provide origin authentication. Finally, privacy protection should be taken into account to limit access to valuable information -- such as identifiers, type of collected data, traffic patterns -- in potential translation points (proxies) or in the backend. The last security suite (SecProf\_4) further extends the previous set of requirements considering security mechanisms to deal with translations between TLS and DTLS or for the provision of secure multicast within a 6LoWPAN/CoAP network from the backend.

## 7. Next Steps towards a Flexible and Secure Internet of Things

This Internet Draft included an overview of both operational and security requirements of things in the Internet of Things, discussed a general threat model and security issues, and introduced a number of potential security suites fitting different types of IoT deployments.

We conclude this document by giving our assessment of the current status of CoAP security with respect to addressing the IP security challenges we identified, so as to facilitate discussion of next steps towards workable security design concepts suitable for IP-based IoT in the broader community. Hereby, we focus on the employed security protocols and the type of security architecture.

With current status, we refer to the feasibility of realizing secure deployments with existing CoAP protocols and the practicality of creating comprehensive security architectures based on those protocols:

- 1    DTLS has been defined as the basic building block for protecting CoAP. At the time it was first proposed, no DTLS implementation for small, constrained devices was available. In the mean-time, TinyDTLS [TinyDTLS] has been developed offering the first open-source implementation of the protocol for small devices. However, more experience with the protocol is required. In particular, a performance evaluation and comparison should be made with a well-defined set of standard node platforms/networks. The results will help understand the limitations and the benefits of DTLS as well as to give recommended usage scenarios for this security protocol.

- 2    (D)TLS was designed for traditional computer networks and, thus, some of its features may not be optimal for resource-constrained networks. This includes:
  - a    Basic DTLS features that are, in our view, not ideal for resource-constrained devices. For instance, the loss of a message in-flight requires the retransmission of all messages in-flight. On the other hand, if all messages in-flight are transmitted together in a single UDP packet, more resources are required for handling of large buffers. As pointed out in [ID-Hartke] , the number of flights in the DTLS handshake should be reduced, so that a faster setup of a secure channel can be realized. This would definitely improve the performance of DTLS significantly.
  - b    Fragmentation of messages due to smaller MTUs in resource-constrained networks is problematic. This implies that the node must have a large buffer to store all the fragments and subsequently perform re-ordering and reassembly in order to construct the entire DTLS message. The fragmentation of the handshake messages can, e.g., allow for a very simple method to carry out a denial of service attack.
  - c    The completion of the DTLS handshake is based on the successful verification of the Finished message by both client and server. As the Finished message is computed based on the hash of all handshake messages in the correct order, the node must allocate a large buffer to queue all handshake messages.
  - d    DTLS is thought to offer end-to-end security; however, end-to-end security also has to be considered from the point of view of LLN protection, so that end-to-end exchanges can still be verified and the LLN protected from, e.g., DoS attacks.
- 3    Raw public-key in DTLS has been defined as mandatory. However, memory-optimized public-key libraries still require several KB of flash and several hundreds of B of RAM. Although Moore's law still applies and an increase of platform resources is expected, many IoT scenarios are cost-driven, and in many use cases, the same work could be done with symmetric-keys. Thus, a key question is whether the choice for raw public-key is the best one. In addition, using raw public keys rather than certified public keys hard codes identities to public keys, thereby inhibiting public key updates and potentially complicating initial configuration.

- 4    Performance of DTLS from a system perspective should be evaluated involving not just the cryptographic constructs and protocols, but should also include implementation benchmarks for security policies, since these may impact overall system performance and network traffic (an example of this would be policies on the frequency of key updates, which would necessitate securely propagating these to all devices in the network).
- 5    Protection of lower protocol layers is a must in networks of any size to guarantee resistance against routing attacks such as flooding or wormhole attacks. The wireless medium that is used by things to communicate is broadcast in nature and allows anybody on the right frequency to overhear and even inject packets at will. Hence, IP-only security solutions may not suffice in many IoT scenarios. At the time of writing the document, comprehensive methods are either not in place or have not been evaluated yet. This limits the deployment of large-scale systems and makes the secure deployment of large scale networks rather infeasible.
- 6    The term "bootstrapping" has been discussed in many occasions. Although everyone agrees on its importance, finding a good solution applicable to most use cases is rather challenging. While usage of existing methods for network access might partially address bootstrapping in the short-term and facilitate integration with legacy back-end systems, we feel that, in the medium-term, this may lead to too large of an overhead and imposes unnecessary constraints on flexible deployment models. The bootstrapping protocol should be reusable and light-weight to fit with small devices. Such a standard bootstrapping protocol must allow for commissioning of devices from different manufacturers in both centralized and ad-hoc scenarios and facilitate transitions of control amongst devices during the device's and system's lifecycle. Examples of the latter include scenarios that involve hand-over of control, e.g., from a configuration device to an operational management console and involving replacement of such a control device. A key challenge for secure bootstrapping of a device in a centralized architecture is that it is currently not feasible to commission a device when the adjacent devices have not been commissioned yet. In view of the authors, a light-weight approach is still required that allows for the bootstrapping of symmetric-keys and of identities in a certified public-key setting.
- 7    Secure resource discovery has not been discussed so far. However, this issue is currently gaining relevance. The IoT, comprising sensors and actuators, will provide access to many resources to sense and modify the environment. The usage of DNS presents

well-known security issues, while the application of secure DNS may not be feasible on small devices. In general, security issues and solutions related to resource discovery are still unclear.

- 8 A security architecture involves, beyond the basic protocols, many different aspects such as key management and the management of evolving security responsibilities of entities during the lifecycle of a thing. This document discussed a number of security suites and argued that different types of security architectures are required. A flexible IoT security architecture should incorporate the properties of a fully centralized architecture as well as allow devices to be paired together initially without the need for a trusted third party to create ad-hoc security domains comprising a number of nodes. These ad-hoc security domains could then be added later to the Internet via a single, central node or via a collection of nodes (thus, facilitating implementation of a centralized or distributed architecture, respectively). The architecture should also facilitate scenarios, where an operational network may be partitioned or merged, and where hand-over of control functionality of a single device or even of a complete subnetwork may occur over time (if only to facilitate smooth device repair/replacement without the need for a hard "system reboot" or to realize ownership transfer). This would allow the IoT to transparently and effortlessly move from an ad-hoc security domain to a centrally-managed single security domain or a heterogeneous collection of security domains, and vice-versa. However, currently, these features still lack validation in real-life, large-scale deployments.
- 9 Currently, security solutions are layered, in the sense that each layer takes care of its own security needs. This approach fits well with traditional computer networks, but it has some limitations when resource-constrained devices are involved and these devices communicate with more powerful devices in the back-end. We argue that protocols should be more interconnected across layers to ensure efficiency as resource limitations make it challenging to secure (and manage) all layers individually. In this regard, securing only the application layer leaves the network open to attacks, while security focused only at the network or link layer might introduce possible inter-application security threats. Hence, the limited resources of things may require sharing of keying material and common security mechanisms between layers. It is required that the data format of the keying material is standardized to facilitate cross-layer interaction. Additionally, cross-layer concepts should be considered for an IoT-driven re-design of Internet security

protocols.

## 8. Security Considerations

This document reflects upon the requirements and challenges of the security architectural framework for Internet of Things.

## 9. IANA Considerations

This document contains no request to IANA.

## 10. Acknowledgements

We gratefully acknowledge feedback and fruitful discussion with Tobias Heer and Robert Moskowitz.

## 11. References

### 11.1. Informative References

[RFC6568]Kim, E., Kaspar, D., and JP. Vasseur, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6568, April 2012.

[RFC2818]Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC6345]Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., Ed., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", RFC 6345, August 2011.

[ID-CoAP]Z. Shelby, K. Hartke, C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18, June 2013.

[ID-CoAPMulticast]Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-12 (work in progress), July 2013.

[ID-Daniel]Park, S., Kim, K., Haddad, W., Chakrabarti, S., and J. Laganier, "IPv6 over Low Power WPAN Security Analysis", Internet Draft draft-daniel-6lowpan-security-analysis-05, Mar 2011.

[ID-HIP]Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.

[ID-Hartke]Hartke, K. and O. Bergmann, "Datagram Transport Layer Security in Constrained Environments", draft-hartke-core-codtls-02 (work in progress), July 2012.

[ID-Moskowitz]Moskowitz, R., Heer, T., Jokela, P., and Henderson, T., "Host Identity Protocol Version 2", draft-ietf-hip-rfc5201-bis-13 (work in progress), Sep 2013.

[ID-Nikander]Nikander, P. and J. Melen, "A Bound End-to-End Tunnel(BEET) mode for ESP", draft-nikander-esp-beet-mode-09, Aug 2008.

[ID-OFlynn]O'Flynn, C., Sarikaya, B., Ohba, Y., Cao, Z., and R. Cragie, "Security Bootstrapping of Resource-Constrained Devices", draft-oflynn-core-bootstrapping-03 (work in progress), Nov 2010.

[ID-Tsao]Tsao, T., Alexander, R., Dohler, M., Daza, V., and A. Lozano, "A Security Framework for Routing over Low Power and Lossy Networks", draft-ietf-roll-security-framework-07, Jan 2012.

[ID-Williams]Williams, M. and J. Barrett, "Mobile DTLS", draft-barrett-mobile-dtls-00, Mar 2009.

[ID-proHTTPCoAP]Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Best practices for HTTP-CoAP mapping implementation", draft-castellani-core-http-mapping-07(work in progress), Feb 2013.

[RFC3261]Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3748]Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC3756]Nikander, P., Ed., Kempf, J., and E. Nordmark, "IPv6 Neighbor Discovery (ND) Trust Models and Threats", RFC 3756, May 2004.

[RFC3833]Atkins, D. and R. Austein, "Threat Analysis of the Domain Name System (DNS)", RFC 3833, August 2004.

[RFC4016]Parthasarathy, M., "Protocol for Carrying Authentication and Network Access (PANA) Threat Analysis and Security Requirements", RFC 4016, March 2005.

[RFC5246]Dierks, T. and E. Rescorla, "The Transport Layer Security



(TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC4251]Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4306]Kaufman, C., Ed., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4555]Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)", RFC 4555, June 2006.

[RFC4621]Kivinen, T. and H. Tschofenig, "Design of the IKEv2 Mobility and Multihoming (MOBIKE) Protocol", RFC 4621, August 2006.

[RFC4738]Ignjatic, D., Dondeti, L., Audet, F., and P. Lin, "MIKEY-RSA-R: An Additional Mode of Key Distribution in Multimedia Internet KEYing (MIKEY)", RFC 4738, November 2006.

[RFC4919]Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.

[RFC4944]Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC5191]Forsberg, D., Ohba, Y., Ed., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.

[RFC5201]Moskowitz, R., Nikander, P., Jokela, P., Ed., and T. Henderson, "Host Identity Protocol", RFC 5201, April 2008.

[RFC5206]Nikander, P., Henderson, T., Ed., Vogt, C., and J. Arkko, "End-Host Mobility and Multihoming with the Host Identity Protocol", RFC 5206, April 2008.

[RFC5238]Phelan, T., "Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)", RFC 5238, May 2008.

[RFC5246]Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5713]Moustafa, H., Tschofenig, H., and S. De Cnodder, "Security Threats and Security Requirements for the Access Node Control Protocol (ANCP)", RFC 5713, January 2010.

[RFC5903]Fu, D. and J. Solinas, "Elliptic Curve Groups modulo a Prime

(ECP Groups) for IKE and IKEv2", RFC 5903, June 2010.

[AUTO-ID]"AUTO-ID LABS", Web <http://www.autoidlabs.org/>, Sept 2010.

[BACNET]"BACnet", Web <http://www.bacnet.org/>, Feb 2011.

[DALI]"DALI", Web <http://www.dalibydesign.us/dali.html>, Feb 2011.

[JOURNAL-Perrig]Perrig, A., Szewczyk, R., Wen, V., Culler, D., and J. Tygar, "SPINS: Security protocols for Sensor Networks", Journal Wireless Networks, Sept 2002.

[NIST]Dworkin, M., "NIST Specification Publication 800-38B", 2005.

[PROC-Chan]Chan, H., Perrig, A., and D. Song, "Random Key Predistribution Schemes for Sensor Networks", Proceedings IEEE Symposium on Security and Privacy, 2003.

[PROC-Gupta]Gupta, V., Wurm, M., Zhu, Y., Millard, M., Fung, S., Gura, N., Eberle, H., and S. Shantz, "Sizzle: A Standards-based End-to-End Security Architecture for the Embedded Internet", Proceedings Pervasive Computing and Communications (PerCom), 2005.

[PROC-Smetters-02]Balfanz, D., Smetters, D., Steward, P., and H. Chi Wong, "Talking To Strangers: Authentication in Ad-Hoc Wireless Networks", Paper NDSS, 2002.

[PROC-Smetters-04]Balfanz, D., Durfee, G., Grinter, R., Smetters, D., and P. Steward, "Network-in-a-Box: How to Set Up a Secure Wireless Network in Under a Minute", Paper USENIX, 2004.

[PROC-Stajano-99]Stajano, F. and R. Anderson, "Resurrecting Duckling - Security Issues for Adhoc Wireless Networks", 7th International Workshop Proceedings, Nov 1999.

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

[THESIS-Langheinrich]Langheinrich, M., "Personal Privacy in Ubiquitous Computing", PhD Thesis ETH Zurich, 2005.

[TinyDTLS "TinyDTLS", Web <http://tinydtls.sourceforge.net/>, Feb 2012.

[WG-6LoWPAN]"IETF 6LoWPAN Working Group", Web <http://tools.ietf.org/wg/6lowpan/>, Feb 2011.

[WG-CORE]"IETF Constrained RESTful Environment (CoRE) Working Group",  
Web <https://datatracker.ietf.org/wg/core/charter/>, Feb 2011.

[WG-MSEC]"MSEC Working Group", Web  
<http://datatracker.ietf.org/wg/msec/>.

[ZB]"ZigBee Alliance", Web <http://www.zigbee.org/>, Feb 2011.

Authors' Addresses

Oscar Garcia-Morchon  
Philips Research  
High Tech Campus  
Eindhoven, 5656 AA  
The Netherlands

Email: oscar.garcia@philips.com

Sandeep S. Kumar  
Philips Research  
High Tech Campus  
Eindhoven, 5656 AA  
The Netherlands

Email: sandeep.kumar@philips.com

Sye Loong Keoh  
University of Glasgow Singapore  
Republic PolyTechnic, 9 Woodlands Ave 9  
Singapore 838964  
SG

Email: SyeLoong.Keoh@glasgow.ac.uk

Rene Hummen  
RWTH Aachen University  
Templergraben 55  
Aachen, 52056  
Germany

Email: rene.hummen@cs.rwth-aachen.de

Rene Struik  
Struik Security Consultancy  
Toronto,  
Canada

Email: rstruik.ext@gmail.com

core  
Internet-Draft  
Intended status: Informational  
Expires: January 10, 2013

B. Greevenbosch  
Huawei Technologies  
July 9, 2012

CoAP Minimum Block Time  
draft-greevenbosch-core-block-minimum-time-00

Abstract

This document defines an "MinimumBlockTime" option for CoAP, which can be used to negotiate the minimum time between two subsequent block requests. It can be used for overload and congestion control.

## Note

Discussion and suggestions for improvement are requested, and should be sent to [core@ietf.org](mailto:core@ietf.org).

## Status of this Memo

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

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

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

This Internet-Draft will expire on January 10, 2013.

## Copyright Notice

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

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

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .                  | 4  |
| 2. Requirements notation . . . . .         | 5  |
| 3. Definitions . . . . .                   | 6  |
| 4. The "MinimumBlockTime" option . . . . . | 7  |
| 5. Legacy behaviour . . . . .              | 8  |
| 6. Open issues . . . . .                   | 9  |
| 7. Example . . . . .                       | 10 |
| 8. Security Considerations . . . . .       | 11 |
| 9. IANA Considerations . . . . .           | 12 |
| 10. Acknowledgements . . . . .             | 13 |
| 11. Normative References . . . . .         | 14 |
| Author's Address . . . . .                 | 15 |

## 1. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a RESTful protocol for constrained nodes and networks. In [I-D.ietf-core-block], the block mechanism for block-wise transmission of data is defined.

This document defines a "MinimumBlockTime" option, which can be used to negotiate the minimum time between two subsequent block requests.

Negotiating the minimum time between the block requests can be used to limit the associated traffic, providing a mechanism for congestion control. In addition, it allows very constrained servers to limit the number of requests they receive within a certain time period, preventing them from becoming overloaded.



## 2. 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].

### 3. Definitions

#### Block transaction

The block-wise transmission of a single resource, between a unique client and server pair.

#### Two subsequent requests

In this document, the phrase "two subsequent requests" indicates two requests in a same block transaction, in which one follows the other, without other requests from the same block transaction in between. Notice that the difference between the block numbers of the two subsequent requests does not have to be one, although most often this will be the case.

#### Block time

The time between two subsequent requests in a block transaction.

#### Block speed

The multiplicative inverse of the block time.

## 4. The "MinimumBlockTime" option

| Type | C/E | Name             | Format | Length | Default |
|------|-----|------------------|--------|--------|---------|
| TBD  | E   | MinimumBlockTime | uint   | 0-2B   | 0       |

Table 1: The "MinimumBlockTime" option

The "MinimumBlockTime" option is an elective option, which is used to negotiate the minimum time in milliseconds that a client needs to wait between sending two subsequent block requests.

In the remainder of this section, it is assumed that both the client and the server support the "MinimumBlockTime" option.

If the client includes a "Block1" or "Block2" option in its first request in a block transaction, it SHOULD include the "MinimumBlockTime" option in that first request too. The server SHOULD include the "MinimumBlockTime" option in its first block response.

In a block request, the option's value  $T_C$  indicates the minimum block time in ms that the client can support.

In a block response, the option's value  $T_S$  indicates the minimum block time in ms that the server can support.

The client SHALL wait at least  $T_S$  ms between sending two subsequent block requests.

The following MUST hold:  $T_S \leq T_C$ .

The "MinimumBlockTime" option has a default value 0. A value  $T_S=0$  indicates the server does not put any restrictions on the block speed. A value  $T_C=0$  indicates that the client prefers to send the requests as quickly as possible.

## 5. Legacy behaviour

It is possible that either the client or server does not support the "MinimumBlockTime" option. If the client does not support the option, then obviously it cannot take the server's preference into account. Similarly if the server does not support the option, it cannot use it to restrict the block speed.

In either case, or their combination, the client will choose the block speed as it prefers. This corresponds to the case  $T_S=0$ .

To allow the server to distinguish between a client that supports the "MinimumBlockTime" option but wants to signal  $T_C=0$ , and a client that does not support the "MinimumBlockTime" option, it is RECOMMENDED for the compliant client to include option in the first request in a Block transaction, even when the client wants to signal  $T_C=0$ .

## 6. Open issues

For longer block transactions, there may be value in allowing updates of the block speed during a block transaction. We should consider whether increasing efficiency will justify the extra complexity.

## 7. Example

Figure 1 contains an example of a block transaction with the "MinimumBlockTime" option. The client indicates its supported minimum block time as 200ms. The associated block speed is too high for the server, so the server indicates a minimum block time of 300ms. The client obeys this value for the rest of the transaction.

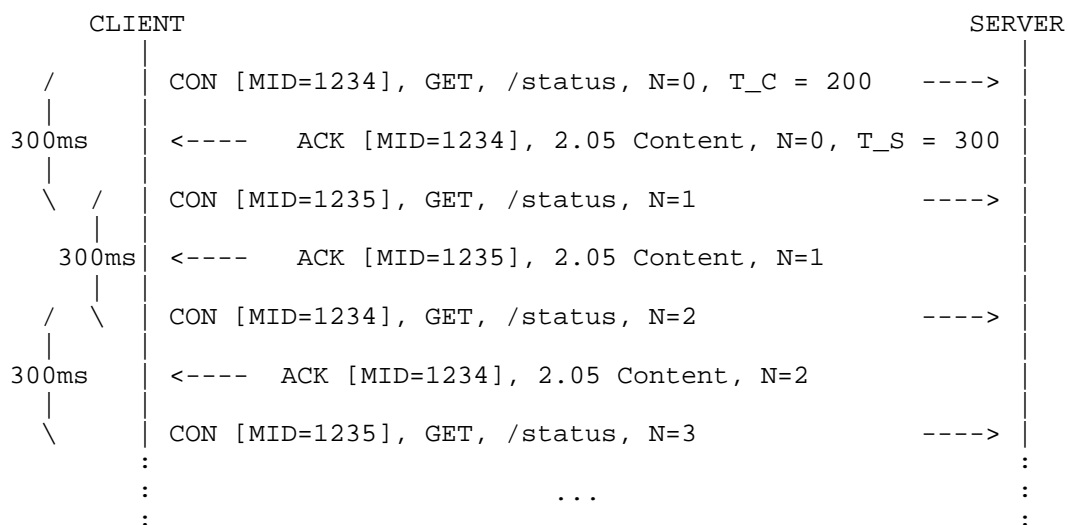


Figure 1: Example of transaction with "MinimumBlockTime"

## 8. Security Considerations

By modifying the value of the "MinimumBlockTime" option to a higher value, a man-in-the-middle could increase the time used to perform a block transaction. When the client encounters a response with a too high "MinimumBlockTime" value, it MAY abort the transaction, and try to reinitiate it. However, to prevent overloading the server, the client MUST limit the number of these reinitiations.

By decreasing the value of the "MinimumBlockTime" option, the man-in-the-middle can induce the client to send block requests at a speed too high for the server. The server should be prepared for this, for example by discarding requests that cannot be processed. This is similar to the case where the server or client does not support the "MinimumBlockTime" option.

## 9. IANA Considerations

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

| Number         | Name             | Reference |
|----------------|------------------|-----------|
| TBD (elective) | MinimumBlockTime | [RFCXXXX] |

Table 2: CoAP option numbers



## 10. Acknowledgements

The author would like to thank Kepeng Li for his ideas and feedback.

## 11. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [I-D.ietf-core-block]  
Shelby, Z. and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-block-08 (work in progress), February 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), March 2012.

Author's Address

Bert Greevenbosch  
Huawei Technologies Co., Ltd.  
Huawei Industrial Base  
Bantian, Longgang District  
Shenzhen 518129  
P.R. China

Phone: +86-755-28978088

Email: bert.greevenbosch@huawei.com



core  
Internet-Draft  
Intended status: Standards Track  
Expires: December 23, 2013

B. Greevenbosch  
Huawei Technologies  
J. Hoebeke  
I. Ishaq  
F. Van den Abeele  
iMinds-IBCN/UGent  
June 21, 2013

CoAP Profile Description Format  
draft-greevenbosch-core-profile-description-02

Abstract

This document provides a profile description format, that can be used to express capabilities of a CoAP server.

Note

Discussion and suggestions for improvement are requested, and should be sent to [core@ietf.org](mailto:core@ietf.org).

Status of This Memo

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

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

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

This Internet-Draft will expire on December 23, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|   |    |
|---|----|
| 1. Requirements notation . . . . .                            | 2  |
| 2. Introduction . . . . .                                     | 2  |
| 3. Obtaining the profile information . . . . .                | 3  |
| 4. The Profile Format . . . . .                               | 4  |
| 4.1. The Path "path" profile field . . . . .                  | 4  |
| 4.2. The Methods "m" profile field . . . . .                  | 4  |
| 4.3. The Options "op" profile field . . . . .                 | 4  |
| 4.4. The Block-Sizes "bls" and "b2s" profile fields . . . . . | 5  |
| 4.5. The Content-Formats "cf" profile field . . . . .         | 5  |
| 5. Usage of URI Queries . . . . .                             | 6  |
| 6. Forward compatibility . . . . .                            | 6  |
| 7. Examples . . . . .   | 6  |
| 8. Open topics . . . . .                                      | 8  |
| 8.1. Open since v00 . . . . .                                 | 8  |
| 8.2. Open since v01 . . . . .                                 | 8  |
| 8.3. Open since v02 . . . . .                                 | 9  |
| 9. Change log . . . . .                                       | 9  |
| 9.1. Changes in v01 . . . . .                                 | 9  |
| 9.2. Changes in v02 . . . . .                                 | 9  |
| 10. Security Considerations . . . . .                         | 9  |
| 11. IANA Considerations . . . . .                             | 9  |
| 12. Acknowledgements . . . . .                                | 10 |
| 13. Normative References . . . . .                            | 10 |
| Authors' Addresses . . . . .                                  | 10 |

## 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].

## 2. Introduction

The Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a RESTful protocol for constrained nodes and networks.

Often, a client first learns about a resource through the link format [I-D.ietf-core-link-format]. The link format only provides basic information, for example the resource URL. However, it would be good if the client could get more extensive information on the resources

when required. This document defines a profile description format, which can be used to signal several parameters about resources and their servers.

One of the main features of the CoAP protocol is the ability to include CoAP options. These options can be either elective or critical. A message with an unsupported critical option will be rejected, whereas unsupported elective options will be ignored.

Even though it is well defined how the server must respond to unsupported options, it is useful for the client to know which options are supported in advance. In this way, it can determine which options are viable to use in a transaction, and which features cannot be exploited. This specification allows signalling of the supported options by the resource.

Another important feature of a CoAP server is which content formats it supports. CoAP provides a mechanism for the client to indicate to the server which content format the client prefers. The use of profiles allows signalling what content formats are supported by the server, so that the client can decide which content type it prefers.

Signalling of the supported CoAP methods (e.g. GET, POST) and maximum block size is also provided. It is anticipated that more profile fields will be defined in the future.

### 3. Obtaining the profile information

Similar to the link format from [RFC6690], the profile interface uses a well-known resource URI as a pointer to the profile description.

The host component of the URI of the profile description should be equal to the URI of the associated resource, whereas the path component begins with ".well-known" as specified in [RFC5785]. The complete path component equals ".well-known/profile".

For example, if the client wants to get the profile description of a server with URI "www.example.org", it can send a GET request for "coap://www.example.org/.well-known/profile". In this case the server SHOULD respond with the profile details of all resources on the server. The server MAY use the reserved resource name "." to provide a default profile. This default profile applies to all resources that are not specifically listed in the profile (e.g. because they do not have their individual profile) and describes the general CoAP capabilities of the server. If a resource has its own profile, then this profile MUST be used and the default profile field MUST be ignored.

If only the profile of a particular resource is needed, the client SHOULD send a GET request including the "path" URI-query. If the resource has no specific profile the server MUST respond with the default profile.

For example, the profile of the sensor "coap://www.example.org/s" can be acquired with a GET to: "coap://www.example.org/.well-known/profile?path=s".

Section 5 covers this in more detail.

#### 4. The Profile Format

The profile description is formatted as a JSON document. It consists of several profile fields, each of which has associated parameters.

##### 4.1. The Path "path" profile field

The "path" profile field contains the Uri-Path component associated with the resource. It can be used to filter on certain profile properties, as described in Section 5.

##### 4.2. The Methods "m" profile field

The methods "m" profile field describes the CoAP methods that the resource supports.

```
"m": [m1, m2, ...]
```

Figure 1: "m" syntax

The methods are defined by integer values containing the method codes as defined in [I-D.ietf-core-coap], Table 5 [TBD update this when CoAP has become RFC.], or registered with IANA.

For simplicity, the "0." prefix is omitted. So in the "m" profile field, GET is indicated by the value 1, POST by 2, PUT by 3 and DELETE by 4.

##### 4.3. The Options "op" profile field

The options "op" profile field contains a list of options that are supported by a resource. It has the format depicted in Figure 2, where op1, op2, ... are integers representing the option numbers of the supported options, as defined in [I-D.ietf-core-coap] or registered with IANA. The option numbers MUST appear in numerical order, starting with the lowest number.



```
"op":[op1,op2,...]
```

Figure 2: "op" syntax

When including the "op" profile field in the profile description of a resource, all option numbers of the CoAP options supported by that resource **MUST** be included. Options that are not supported by the resource **MUST NOT** be included in the "op" profile field.

If the "op" profile field is available, the receiving party **SHALL** assume a non-listed option is not supported by the associated resource.

#### 4.4. The Block-Sizes "bls" and "b2s" profile fields

The block sizes "bls" and "b2s" profile fields indicate which block sizes are supported for Block1 and Block2 options when block-wise transfer is used. It has the format depicted in Figure 3, where bls1, bls2, ... are three-bit unsigned integers indicating the size of a block to the power of two. Thus block size =  $2^{(bl + 4)}$ . The allowed values of bl are 0 to 6, i.e., the minimum block size is  $2^{(0+4)} = 16$  and the maximum is  $2^{(6+4)} = 1024$ . The block-size numbers **MUST** appear in numerical order, starting with the lowest number (see [I-D.ietf-core-block]).

```
"bls":[bls1,bls2,...]  
"b2s":[b2s1,b2s2,...]
```

Figure 3: "bls" and "b2s" syntax

When including the "bls" or the "b2s" profile fields in the profile description of a resource, all respective Block1 and Block2 sizes that are supported in block-wise transfer by that resource **MUST** be included. Block sizes that are not supported by the resource **MUST NOT** be included in the "bls" or the "b2s" profile fields.

If the "bls" or the "b2s" profile fields are available, the receiving party **SHALL** assume a non-listed block size is not supported by the associated resource. If only one of the "bls" and the "b2s" profile fields is available, the receiving party **SHALL** assume that the other block transfer is not supported by the associated resource.

#### 4.5. The Content-Formats "cf" profile field

The content formats "cf" profile field indicates which content formats are supported. It has the format depicted in Figure 4, where cf1, cf2, ... are integers representing the numbers of the supported content formats, as defined in [I-D.ietf-core-coap] or registered

with IANA. The content format numbers MUST appear in numerical order, starting with the lowest number.

```
"cf":[cf1,cf2,...]
```

Figure 4: "cf" syntax

When including the "cf" profile field in the profile description of a resource, all content formats of the CoAP options supported by that resource MUST be included. Content formats that are not supported by the resource MUST NOT be included in the "cf" profile field.

If the "cf" profile field is available, the receiving party SHALL assume a non-listed content format is not supported by the associated resource.

## 5. Usage of URI Queries

To specify which information is needed, the client MAY include an "Uri-Query" option in its request for the profile description. The server SHOULD understand and process this information, although constraint servers MAY omit the functionality. In the latter case, they SHOULD return the same results as if the "Uri-Query" option was not included.

The URI Queries are of the form "N=V", where N is the name of the field to filter on, and V is the desired value.

For example, if the client wants to know all resources on the server that support content format "application/json", which has the number 50 (see [I-D.ietf-core-coap]), then it will include a "Uri-Query" option with the value "cf=50".

When the request contains multiple "Uri-Query" options, "AND" semantics hold.

## 6. Forward compatibility

To allow addition of new profile fields in the future, unknown profile fields SHOULD be ignored by the client.

## 7. Examples

The following is an example of a camera sensor at "coap://www.example.org/cam", that supports the "Uri-Host" (3), "ETag" (4), "Uri-Port" (7), "Uri-Path" (11), "Content-Format" (12), "Token" (19), "Block2" (23) and "Proxy-Uri" (35) options.

The supported content formats are "text/plain" (0), "application/link-format" (40) and "application/json" (50).

The camera only supports the GET method (1).

The camera can support Block2 with Block sizes of 256 or 512 bytes.

Req: GET coap://www.example.org/.well-known/profile

Res: 2.05 Content (application/json)

```
{
  "profile":
  {
    "path": "cam",
    "m": [1],
    "op": [3,4,7,11,12,19,23,35],
    "b2s": [4,5],
    "cf": [0,40,50]
  }
}
```

If the server also supports three other resources, such as a temperature sensor (which can do observe), a humidity and a fire detector, the request/response pair would look as follows:

Req: GET coap://www.example.org/.well-known/profile

Res: 2.05 Content (application/json)

```
{
  "profile": [
    {
      "path": ".",
      "m": [1],
      "op": [3,4,7,11,12,19,35],
      "cf": [0]
    },
    {
      "path": "cam",
      "m": [1],
      "op": [3,4,7,11,12,19,23,35],
      "b2s": [4,5],
      "cf": [0,40,50]
    },
    {
      "path": "temperature",
      "m": [1],
      "op": [3,4,6,7,11,12,19,35],

```

```
        "cf":[0]
      }
    ]
  }
```

Please note that the response did not include profiles for the "fire" and "humidity" resources. Instead it included a default profile that applies for these two not explicitly mentioned resources.

If the client now wants to get the resources that support content-format "application/json" (50) it looks as follows:

Req: GET coap://www.example.org/.well-known/profile?cf=50

Res: 2.05 Content (application/json)

```
{
  "profile":
  {
    "path":"cam",
    "m":[1],
    "op":[3,4,7,11,12,19,23,35],
    "b2s":[4,5],
    "cf":[0,40,50]
  }
}
```

## 8. Open topics

### 8.1. Open since v00

- o How to signal the client profile?
- o Which other profile data needs signalling?
- o A natural content format for a camera would be JPEG. Therefore the "image/jpeg" content format may need CoAP registration.
- o Fix the order in which the profile fields must appear?

### 8.2. Open since v01

- o For the time being, text about the hierarchy of profiles in servers, batches and resources has been removed. This leads to a requirement to provide the profile description for each separate resource. A mechanism to re-introduce hierarchy may make significantly reduce the profile description verbosity.

### 8.3. Open since v02

- o The "cf" field contains redundant information, as supported content types can also be signalled through the link format "ct" attribute. However, it could be useful to insert this information in ".well-known/profile" too, such that a receiving entity does not need to gather the information from two sources.

## 9. Change log

### 9.1. Changes in v01

- o Changed from /p suffix to usage of ".well-known/profile"
- o Added support of Uri-Query
- o Updated option numbering according to [I-D.ietf-core-coap]
- o Changed Media Type and "mt" to Content Format and "cf", in accordance with [I-D.ietf-core-coap]
- o Expanded examples
- o Removed text about the hierarchy
- o Added default profile "."
- o Added "bls" and "b2s" fields for block size

### 9.2. Changes in v02

- o Added methods "m" profile field.
- o The order of the fields has been changed in both the description and the examples. This order is not mandated though.

## 10. Security Considerations

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

In an unprotected environment, an attacker can change the profile description. 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. However, such a threat is normal in environments that lack secure authentication.

## 11. IANA Considerations

- o A registry for profile fields as well as possible values needs to be set up.
- o The ".well-known/profile" path component must be registered.

## 12. Acknowledgements

The authors would like to thank Kepeng Li for his ideas and feedback.

## 13. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-11 (work in progress), March 2013.
- [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.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-14 (work in progress), June 2012.

## Authors' Addresses

Bert Greevenbosch  
Huawei Technologies Co., Ltd.  
Huawei Industrial Base  
Bantian, Longgang District  
Shenzhen 518129  
P.R. China  
  
Phone: +86-755-28978088  
Email: bert.greevenbosch@huawei.com

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

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

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

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

CoRE Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 17, 2013

K. Hartke  
O. Bergmann  
Universitaet Bremen TZI  
July 16, 2012

Datagram Transport Layer Security in Constrained Environments  
draft-hartke-core-codtls-02

Abstract

This draft considers some obstacles in implementing Datagram Transport Layer Security (DTLS) in constrained environments, and presents some ideas for a constrained version of DTLS that is friendly to constrained nodes and networks.

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 January 17, 2013.

Copyright Notice

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

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



## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .                              | 3  |
| 1.1. Background . . . . .                              | 3  |
| 1.2. Overview . . . . .                                | 3  |
| 1.3. Terminology . . . . .                             | 4  |
| 2. Potential Problems and Possible Solutions . . . . . | 4  |
| 2.1. Handshake Message Fragmentation . . . . .         | 4  |
| 2.2. Timer Values . . . . .                            | 6  |
| 2.3. Connection Initiation . . . . .                   | 7  |
| 2.4. Connection Closure . . . . .                      | 9  |
| 2.5. Application Data Fragmentation . . . . .          | 9  |
| 2.6. Data size . . . . .                               | 10 |
| 2.7. Code size . . . . .                               | 11 |
| 3. Stateless Header Compression . . . . .              | 12 |
| 3.1. Records . . . . .                                 | 12 |
| 3.2. Handshake Messages . . . . .                      | 13 |
| 4. RESTful DTLS Handshake . . . . .                    | 15 |
| 5. Security Considerations . . . . .                   | 17 |
| 6. IANA Considerations . . . . .                       | 17 |
| 7. Acknowledgements . . . . .                          | 17 |
| 8. References . . . . .                                | 17 |
| 8.1. Normative References . . . . .                    | 17 |
| 8.2. Informative References . . . . .                  | 17 |
| Appendix A. Templates . . . . .                        | 20 |
| A.1. secp256r1 . . . . .                               | 20 |
| A.2. secp384r1 . . . . .                               | 21 |
| A.3. secp521r1 . . . . .                               | 22 |
| Authors' Addresses . . . . .                           | 23 |

## 1. Introduction

### 1.1. Background

Nodes that take part in the "Internet of Things" often have strict limitations regarding their computational power, memory size (both ROM and RAM), and power management [I-D.ietf-lwig-guidance]. Network communication, especially wireless, also imposes constraints that need to be considered during protocol design, e.g. low bitrate, variable delay and possibly high packet loss. Moreover, frames at the link layer might be much smaller than the IPv6 minimum MTU of 1280 bytes and therefore require additional mapping mechanisms such as 6LoWPAN [RFC4944] for IEEE 802.15.4 wireless networks [IEEE.802-15-4], which in turn may exacerbate the limitations of the network: E.g., as high loss rates are anticipated by design, application protocols usually try to avoid fragmentation at the network layer.

However, application protocols often delegate security mechanisms to transport layer security protocols. More often than not, the protocol overhead from securing the communication is highly relevant to the overall performance of the systems.

One protocol that has received significant attention recently for constrained node/network applications is Datagram Transport Layer Security (DTLS) [RFC6347]. DTLS is derived from and inherits some characteristics from TLS [RFC5246]. Although DTLS has not been designed with constrained nodes/networks in mind, it is thought to be usable in such environments [SOS12]. Still, there are a few challenges when it comes to implement DTLS.

### 1.2. Overview

The present document considers some obstacles in implementing DTLS in constrained environments, and presents a few ideas to make DTLS more friendly to constrained nodes and networks.

The ideas generally fall into one of the following categories:

Implementation guidance: Implementation techniques for achieving light-weight implementations of DTLS, without affecting conformance to the relevant specifications or interoperability with other implementations. This includes techniques for reducing complexity, memory footprint, or power usage. The result may eventually be incorporated into [I-D.ietf-lwig-guidance].

Protocol profile: Use of DTLS in a particular way, for example, by changing MAYs into MUSTs or MUST NOTs, or by requiring or precluding certain extensions or cipher suites. Existing DTLS implementations ought to continue to be used without change if they can be configured accordingly.

Stateless header compression: Compression of DTLS records without explicitly building any compression context state. This is done by using shorter forms to represent the same bits of information or relying on information that is already shared by the client and server. Existing DTLS implementations can continue to be used if a thin layer is added that handles compression/decompression.

Breaking changes: New implementations are required that do not interoperate with implementations of DTLS.

### 1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. (Note that this document itself is informational, but it is discussing normative statements.)

The term "byte" is used in its now customary sense as a synonym for "octet".

## 2. Potential Problems and Possible Solutions

### 2.1. Handshake Message Fragmentation

DTLS records can be large in size for a single 6LoWPAN [RFC4944] payload: [IEEE.802-15-4] specifies a physical layer MTU of only 127 bytes, which yields about 60-80 bytes of payload after adding MAC layer and adaptation layer headers. Although 6LoWPAN supports the fragmentation of IPv6 packets into small link-layer frames, this doesn't really work well for constrained applications and networks.

DTLS offers fragmentation at the handshake layer and hence can get around IP fragmentation. However, this can add a significant overhead on the number of datagrams and bytes transferred (see Table 1). Packet loss is also still a big problem for the constrained nodes; buffers must be large enough to hold all messages after reassembly and losing a single fragment will cause all fragments of a message flight to be retransmitted. This is very likely especially during key and certificate exchange as these will not fit within a packet without fragmentation in most 6LoWPANs.

| UDP data<br>size limit<br>(bytes) | Number of<br>datagrams<br>transferred | Total number of<br>bytes<br>transferred | Proportion of<br>header data |
|-----------------------------------|---------------------------------------|---|------------------------------|
| 50                                | 27                                    | 1,182                                   | 55 %                         |
| 55                                | 21                                    | 1,037                                   | 49 %                         |
| 60                                | 20                                    | 1,081                                   | 51 %                         |
| 65                                | 18                                    | 1,003                                   | 47 %                         |
| 70                                | 15                                    | 912                                     | 42 %                         |
| 75                                | 14                                    | 875                                     | 39 %                         |
| 80                                | 13                                    | 874                                     | 39 %                         |
| 85                                | 12                                    | 849                                     | 37 %                         |
| 90                                | 12                                    | 849                                     | 37 %                         |
| 1,152                             | 6                                     | 802                                     | 34 %                         |

Table 1: Number of datagrams and bytes transferred using different limits for DTLS fragmentation in an example DTLS handshake (TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CCM\_8 with raw public key certificate)

Possible Solutions include:

- o Use IP fragmentation instead of DTLS fragmentation. If no X.509 certificates are involved, the handshake messages of one flight typically require less than 400 bytes combined. Since all messages of a flight are retransmitted anyway when a single fragment is lost, the difference between performing the fragmentation at the DTLS layer and at the IP layer is probably not huge. A recipient must still be prepared to receive arbitrarily fragmented handshake messages at the DTLS layer, though.
- o Reduce the number of bytes to be transferred, so the overhead of header data becomes smaller when fragmenting for small packet sizes, and fewer packets need to be transmitted that could potentially be lost:
  - \* Use an out-of-band mechanism to exchange large blobs. For example, the TLS Cached Information Extension [I-D.ietf-tls-cached-info] allows to omit the exchange of fairly static information, such as the server certificate, if this information is already available.
  - \* Use 6LoWPAN General Header Compression [I-D.bormann-6lowpan-ghc] to compress DTLS messages, as proposed in [DCOSS12].

- \* Use some DTLS-specific kind of Stateless Header Compression, as shown in Section 3. This can significantly reduce the number of datagrams and bytes transferred, and in particular also the proportion of header data in the number of bytes transferred (see Table 2).
  - \* Use compressed point formats for elliptic curve points.
  - \* Use self-delimiting numeric values [RFC6256] instead of fixed-size numeric values.
  - \* Use a bit field instead of multiple type fields to indicate which handshake messages are present in a datagram.
- o Perform the DTLS handshake over another protocol, for example, CoAP [I-D.ietf-core-coap] with its support for block-wise transfers [I-D.ietf-core-block], as shown in Section 4.

| UDP data<br>size limit<br>(bytes) | Number of<br>datagrams<br>transferred | Total number of<br>bytes<br>transferred | Proportion of<br>header data |
|-----------------------------------|---------------------------------------|---|------------------------------|
| 50                                | 15 (56 %)                             | 592 (50 %)                              | 10 %                         |
| 55                                | 13 (62 %)                             | 585 (56 %)                              | 9 %                          |
| 60                                | 13 (65 %)                             | 621 (57 %)                              | 14 %                         |
| 65                                | 11 (61 %)                             | 588 (59 %)                              | 10 %                         |
| 70                                | 11 (73 %)                             | 573 (63 %)                              | 7 %                          |
| 75                                | 11 (79 %)                             | 573 (65 %)                              | 7 %                          |
| 80                                | 10 (77 %)                             | 567 (65 %)                              | 6 %                          |
| 85                                | 10 (83 %)                             | 567 (67 %)                              | 6 %                          |
| 90                                | 10 (83 %)                             | 567 (67 %)                              | 6 %                          |
| 1,152                             | 6 (100 %)                             | 617 (77 %)                              | 14 %                         |

Table 2: Number of datagrams and bytes transferred in the same example DTLS handshake but using Stateless Header Compression (Section 4)

## 2.2. Timer Values

DTLS leaves the choice of timer values to the implementation, but makes the following recommendation:

"Implementations SHOULD use an initial timer value of 1 second (the minimum defined in RFC 6298 [RFC6298]) and double the value at each retransmission, up to no less than the RFC 6298 maximum of 60 seconds." [RFC6347]

Given the time required by some algorithms when executed on a constrained devices (see Table 3), an initial value of 1 second can easily lead to spurious retransmissions.

| Algorithm   | Library      | Memory footprint (bytes) | Execution time (seconds) | Comparable RSA key length |
|-------------|--------------|--------------------------|--------------------------|---------------------------|
| RSA 1024    | AvrCryptolib | 640                      | 199.7                    |                           |
| RSA 2048    | AvrCryptolib | 1,280                    | 1,587.6                  |                           |
| ECDSA 160r1 | TinyECC      | 892                      | 2.3                      | 1024                      |
| ECDSA 192r1 | TinyECC      | 1,008                    | 3.6                      | 1536                      |
| ECDSA 160r1 | Wiselib      | 842                      | 20.2                     | 1024                      |
| ECDSA 192r1 | Wiselib      | 952                      | 34.6                     | 1536                      |
| ECDSA 163k1 | Relic        | 2,804                    | 0.3                      | 1024                      |
| ECDSA 233k1 | Relic        | 3,675                    | 1.8                      | 2048                      |

Table 3: RSA private key operation and ECDSA signature performance (from [I-D.aks-crypto-sensors])

Possible Solutions include:

- o Adjust the timer value to meet the conditions of constrained nodes and low-power, lossy networks.
- o Add some kind of acknowledgment message to DTLS that allows an implementation to confirm the receipt of a message before preparing the next message flight.

### 2.3. Connection Initiation

Nodes with very constrained main memory also suffer from the complexity of the DTLS handshake protocol. We envision that the acceptance of DTLS as security protocol for embedded devices would significantly increase if a less complex connection initiation procedure with a smaller number of handshake messages was defined.

Compared to TLS, DTLS exacerbates the connection initiation: A DTLS handshake has an additional roundtrip that results from the addition of a stateless cookie exchange. This exchange is designed to prevent certain denial-of-service attacks: consumption of excessive server resources caused by the transmission of a series of handshake initiation requests, and use of the server as an amplifier by sending connection initiation messages with a forged source of the victim.

Possible Solutions include:

- o Create the DTLS connection before it is needed, so it doesn't take a long time to set it up when it's actually needed. This works if a server has to deal with a relatively small overall number of clients that wish to interact with the server. Care must be taken such that not all clients perform their handshake at the same time, as a handshake requires considerably more memory than keeping a connection open. (See also Section 2.4 below.)
- o Shorten the handshake to four flights. This may be possible without losing the denial-of-service roundtrip if the cipher suite permits that the server remains stateless after sending the ServerHello and if the flight fits in one datagram (see Figure 1).

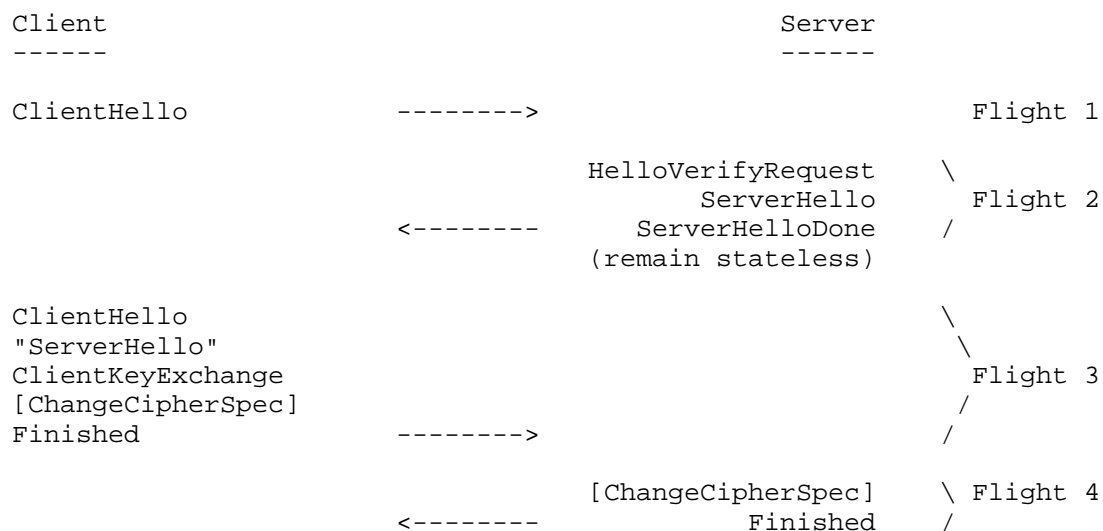


Figure 1: Artist's impression of a four-flight DTLS handshake with Pre-Shared Key

- o As an alternative, client puzzles could be used as a mechanism for mitigating denial-of-service attacks, resulting in a four-flight exchange similar to the one in HIP DEX [I-D.moskowitz-hip-rg-dex]. The application of client puzzles to TLS has been shown [USENIX01]. However, a puzzle would be needed that ideally takes less effort for a constrained device and more effort for a less constrained device.

## 2.4. Connection Closure

Although a connection needs considerably less memory after a handshake has finished, it still requires, e.g., around 80 bytes with AES-128-CCM [I-D.mcgregw-tls-aes-ccm] for the keys, sequence numbers and anti-replay window. More memory is needed if session resumption is supported, to keep the 48-byte master secret and negotiated connection parameters. This limits how many connections a constrained device can maintain at a given time. Often, constrained devices will have a fixed number of "slots" for connections rather than allocating memory dynamically for each connection.

DTLS provides a facility for secure connection closure. When a valid closure alert is received, an implementation can be assured that no further data will be received on that connection. It is noteworthy, though, that the closure alert is not a handshake message and thus is not retransmitted when packet loss occurs.

Possible Solutions include:

- o Maintain the session for as long as possible. When the server runs out of resources, it can close connections, e.g., using a Least Frequently Used (LFU) eviction policy. The client simply assumes that the connection is active until the server rejects its application data, in which case the client initiates a new connection.
- o Use the DTLS Heartbeat Extension [RFC6520] to figure out from time to time if the connection is still active.

## 2.5. Application Data Fragmentation

Messages larger than an IP fragment result in undesired packet fragmentation. DTLS does not support fragmentation of application data. If an implementation of an application layer protocol such as CoAP [I-D.ietf-core-coap] wants to avoid IP fragmentation, it must fit the application data (e.g., a CoAP message) and all headers within a single IP packet.

DTLS has a per-record overhead of 13 bytes for the record header. AEAD ciphers such as AES-CCM [I-D.mcgregw-tls-aes-ccm] eat up additional space to carry the explicit nonce and the authentication tag. Thus, cipher suites like TLS\_PSK\_WITH\_AES\_128\_CCM\_8 or TLS\_ECDHE\_ECDSA\_AES\_128\_CCM\_8 requires 16 additional bytes, leading to an overall overhead of 29 bytes for the header of each encrypted DTLS packet. With packet sizes of 60-80 bytes, this takes a considerable portion of the available packet size away (see Table 4).



| UDP data size<br>limit (bytes) | Number of bytes left<br>for application data | ... with Stateless<br>Header Compression |
|--------------------------------|--|--|
| 50                             | 21 (42 %)                                    | 39 (78 %)                                |
| 55                             | 26 (47 %)                                    | 44 (80 %)                                |
| 60                             | 31 (52 %)                                    | 49 (82 %)                                |
| 65                             | 36 (55 %)                                    | 54 (83 %)                                |
| 70                             | 41 (59 %)                                    | 59 (84 %)                                |
| 75                             | 46 (61 %)                                    | 64 (85 %)                                |
| 80                             | 51 (64 %)                                    | 69 (86 %)                                |
| 85                             | 56 (66 %)                                    | 74 (87 %)                                |
| 90                             | 61 (68 %)                                    | 79 (88 %)                                |
| 1,152                          | 1,123 (97 %)                                 | 1,141 (99 %)                             |

Table 4: Number of bytes left for data in an ApplicationData record using DTLS and DTLS with Stateless Header Compression (Section 4)

Possible Solutions include:

- o Elide the GenericAEADCipher.nonce\_explicit field when AES-CCM is used. The GenericAEADCipher.nonce\_explicit field is set to the 16-bit epoch concatenated with the 48-bit sequence number, which means that the epoch and sequence number are unnecessarily included twice in each record.
- o Elide the DTLS version field where it is implicitly clear. Since the DTLS version is negotiated in the handshake, there should not be a need to specify the DTLS version in each and every record.
- o Elide the length field of the last record in a datagram. DTLS records specify their length so multiple records can be transmitted in a single datagram. When DTLS is used with UDP (which preserves the boundaries of all message sent), the length field of the last record in a datagram can be calculated from the UDP payload length.

For example, when using the Stateless Header Compression presented in Section 3 and eliminating the redundant epoch and sequence number information, the number of bytes left in an ApplicationData record for application data can be significantly increased (see Table 4).

## 2.6. Data size

As fragmented handshake messages can arrive at a constrained node in any order, the receiver must provide a message buffer that is large enough to hold multiple fragments. When several handshake messages

forming a single flight are sent out in parallel, it is likely that the receiver's resources are too limited to order fragments from distinct handshake messages. Avoiding this might require additional resources on the server side to ensure serialization of a flight's messages.

Furthermore, since handshake messages can be fragmented arbitrarily and with overlaps, the receiver must, in addition to the message buffer, keep track of the fragments received so far.

Possible retransmissions require even more buffer space as replay-protection requires encryption of every single packet that is to be transmitted. In particular, this renders destructive in-place encryption impossible as the source data must be preserved.

Possible Solutions include:

- o Use the same sequence number when retransmitting a message, so the plaintext could be encrypted in-place without the need for a second buffer. The security implications of this change need to be carefully analyzed.
- o Favour cryptographic algorithms that use less memory, possibly resulting in a slower performance.
- o Add some kind of acknowledgment message to DTLS that allows an receiver to confirm the receipt of a message, and let the sender wait for the acknowledgment before it sends the next part of the flight.

## 2.7. Code size

Although probably not as severe as data size limits, the code size of a DTLS implementation also can play a role, in particular for constrained devices at the lower bound of Class 1 devices.

Possible Solutions include:

- o Avoid static tables for cryptographic functions where possible, as typical embedded platforms are more restricted in RAM than in non-volatile memory such as flash ROM. Instead, their procedural equivalent is to be used, although less efficient during run-time.
- o Use using pre-composed messages instead of writing code, e.g., for encoding or decoding ASN.1 structures, as shown in Appendix A.

### 3. Stateless Header Compression

Stateless Header Compression compresses the headers of records and handshake messages. The compression is lossless, does not increase the record length and is done without explicitly building any compression context state.

The Finished MAC is computed as if each handshake message had been sent uncompressed.

#### 3.1. Records

Records are compressed by specifying the type, version, epoch, sequence\_number and length fields using a variable number of bytes. A prefix is added in front of the structure to indicate the length of each field or to specify the value of the field directly. If the value is specified directly, the field itself is elided. The format of the prefix is as follows:

```

      0                               1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
      +---+---+---+---+---+---+---+---+
      |0| T | V |   E   |1 1 0| S | L |
      +---+---+---+---+---+---+---+---+

```

The fields in the prefix are defined as follows:

T: Describes the type field.

- 0 - Content Type 20 (ChangeCipherSpec)
- 1 - 8-bit type field
- 2 - Content Type 22 (Handshake)
- 3 - Content Type 23 (Application Data)

V: Describes the version field.

- 0 - Version 254.255 (DTLS 1.0)
- 1 - 16-bit version field
- 2 - Version 254.253 (DTLS 1.2)
- 3 - Reserved for future use

E: Describes the epoch field.

- 0 - Epoch 0
- 1 - Epoch 1
- 2 - Epoch 2
- 3 - Epoch 3
- 4 - Epoch 4

- 5 - 8-bit epoch field
- 6 - 16-bit epoch field
- 7 - Implicit -- same as previous record in the datagram

S: Describes the sequence\_number field.

- 0 - Sequence number 0
- 1 - 8-bit sequence\_number field
- 2 - 16-bit sequence\_number field
- 3 - 24-bit sequence\_number field
- 4 - 32-bit sequence\_number field
- 5 - 40-bit sequence\_number field
- 6 - 48-bit sequence\_number field
- 7 - Implicit -- number of previous record in the datagram + 1

L: Describes the length field.

- 0 - Length 0
- 1 - 8-bit length field
- 2 - 16-bit length field
- 3 - Implicit -- last record in the datagram

### 3.2. Handshake Messages

Handshake messages are compressed in a similar way. A prefix is added in front of the structure to indicate the length of each field or to specify the value of the field directly. If the value is specified directly, the field itself is elided. The format of the prefix is as follows:

|                                     |   |
|-------------------------------------|---|
| 0                                   | 1 |
| 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5     |   |
| +-----+                             |   |
| 0 0      T      L      S      O   C |   |
| +-----+                             |   |

The fields in the prefix are defined as follows:

T: Describes the msg\_type field.

- 0 - 8-bit msg\_type field
- 1 - Handshake Type 1 (Client Hello)
- 2 - Handshake Type 2 (Server Hello)
- 3 - Handshake Type 3 (Hello Verify Request)
- 4 - Reserved for future use
- 5 - Reserved for future use
- 6 - Reserved for future use
- 7 - Handshake Type 11 (Certificate)

- 8 - Handshake Type 12 (Server Key Exchange)
- 9 - Handshake Type 13 (Certificate Request)
- 10 - Handshake Type 14 (Server Hello Done)
- 11 - Handshake Type 15 (Certificate Verify)
- 12 - Handshake Type 16 (Client Key Exchange)
- 13 - Reserved for future use
- 14 - Reserved for future use
- 15 - Handshake Type 20 (Finished)

L: Describes the length field.

- 0 - Implicit -- last message in the record
- 1 - 8-bit length field
- 2 - 16-bit length field
- 3 - 24-bit length field

S: Describes the message\_seq field.

- 0 - Message sequence number 0
- 1 - Message sequence number 1
- 2 - Message sequence number 2
- 3 - Message sequence number 3
- 4 - Message sequence number 4
- 5 - Message sequence number 5
- 6 - Message sequence number 6
- 7 - Message sequence number 7
- 8 - Message sequence number 8
- 9 - Message sequence number 9
- 10 - Message sequence number 10
- 11 - Message sequence number 11
- 12 - Message sequence number 12
- 13 - 8-bit message\_seq field
- 14 - 16-bit message\_seq field
- 15 - Implicit -- number of previous message in the record + 1

O: Describes the fragment\_offset field.

- 0 - Offset 0
- 1 - 8-bit fragment\_offset field
- 2 - 16-bit fragment\_offset field
- 3 - 24-bit fragment\_offset field

C: Describes the fragment\_length field.

- 0 - Implicit -- last message in the record
- 1 - 8-bit fragment\_length field
- 2 - 16-bit fragment\_length field
- 3 - 24-bit fragment\_length field

#### 4. RESTful DTLS Handshake

Where DTLS is used in conjunct with the Constrained Application Protocol (CoAP) [I-D.ietf-core-coap], it might be beneficial to use CoAP with its support for block-wise transfers [I-D.ietf-core-block] instead of DTLS's convoluted handshake protocol to transport DTLS handshake messages.

CoAP, like HTTP, is designed for applications following the REST architectural style [REST]. So the DTLS connection is modeled as a CoAP resource which gets created when a client wants to initiate a connection, and gets updated to modify the state and parameters of the connection. A well-known URI path [RFC5785] is used to identify a collection resource that models the set of active connections and allows new connections to be created.

| Client   |        | Server  |
|--|--------|---|
| -----  |        | -----   |
| POST /.well-known/dtls<br>ClientHello  | -----> |   |
|  | <----- | 1.xx Verify<br>HelloVerifyRequest   |
| POST /.well-known/dtls<br>ClientHello  | -----> |   |
|  | <----- | 2.01 Created /session/4ad6bc29<br>ServerHello<br>Certificate*<br>ServerKeyExchange*<br>CertificateRequest*<br>ServerHelloDone |
| PATCH /session/4ad6bc29<br>Certificate*<br>ClientKeyExchange<br>CertificateVerify*<br>[ChangeCipherSpec]<br>Finished | -----> |   |
|  | <----- | 2.04 Changed<br>[ChangeCipherSpec]<br>Finished  |

Figure 2: Message Flights for Full Handshake

| Client                  |        | Server             |
|-------------------------|--------|--------------------|
| -----                   |        | -----              |
| PATCH /session/4ad6bc29 |        |                    |
| ClientHello             | -----> |                    |
|                         |        | 2.04 Changed       |
|                         |        | ServerHello        |
|                         |        | [ChangeCipherSpec] |
|                         | <----- | Finished           |
| PATCH /session/4ad6bc29 |        |                    |
| [ChangeCipherSpec]      |        |                    |
| Finished                | -----> |                    |
|                         | <----- | 2.04 Changed       |

Figure 3: Message Flights for Session-Resuming Handshake

There are the following possible operations:

- o POST to well-known URI: requests the server to create a new session resource.
- o PATCH session resource: requests the server to change session parameters, or to resume a session.
- o GET session resource: returns a representation of the session.
- o DELETE session resource: requests the server to delete the session resource and free all resources related to the session.

The following protocols and URI schemes are used:

- o CoAP [I-D.ietf-core-coap]
- o coap+codtls://

The following well-known URIs are used:

- o /.well-known/dtls

The following media types are used:

- o application/dtls

(The exact definition of these items is TBD.)

## 5. Security Considerations

Beyond stateless header compression and profiling, changes to the TLS/DTLS protocol need to be performed extremely carefully. No analysis has been done in the present version of this draft.

## 6. IANA Considerations

This draft includes no request to IANA.

## 7. Acknowledgements

Thanks to Angelo P. Castellani, Stefan Jucker and Shahid Raza for helpful comments and discussions that have shaped the document.

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

### 8.2. Informative References

- [DCOSS12] Raza, S., Trabalza, D., and T. Voigt, "6LoWPAN Compressed DTLS for CoAP", 8th IEEE International Conference on Distributed Computing in Sensor Systems, May 2012.
- [I-D.aks-crypto-sensors] Sethi, M., Arkko, J., Keranen, A., and H. Rissanen, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks", draft-aks-crypto-sensors-02 (work in progress), March 2012.
- [I-D.bormann-6lowpan-ghc] Bormann, C., "6LoWPAN Generic Compression of Headers and Header-like Payloads", draft-bormann-6lowpan-ghc-04 (work in progress), March 2012.



- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP",  
draft-ietf-core-block-08 (work in progress),  
February 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.ietf-lwig-guidance]  
Bormann, C., "Guidance for Light-Weight Implementations of  
the Internet Protocol Suite", draft-ietf-lwig-guidance-01  
(work in progress), July 2012.
- [I-D.ietf-tls-cached-info]  
Santesson, S. and H. Tschofenig, "Transport Layer Security  
(TLS) Cached Information Extension",  
draft-ietf-tls-cached-info-11 (work in progress),  
December 2011.
- [I-D.ietf-tls-oob-pubkey]  
Wouters, P., Gilmore, J., Weiler, S., Kivinen, T., and H.  
Tschofenig, "TLS Out-of-Band Public Key Validation",  
draft-ietf-tls-oob-pubkey-03 (work in progress),  
April 2012.
- [I-D.mcgrew-tls-aes-ccm]  
McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for TLS",  
draft-mcgrew-tls-aes-ccm-03 (work in progress),  
February 2012.
- [I-D.mcgrew-tls-aes-ccm-ecc]  
McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-  
CCM ECC Cipher Suites for TLS",  
draft-mcgrew-tls-aes-ccm-ecc-02 (work in progress),  
October 2011.
- [I-D.moskowitz-hip-rg-dex]  
Moskowitz, R., "HIP Diet EXchange (DEX)",  
draft-moskowitz-hip-rg-dex-06 (work in progress),  
May 2012.
- [IEEE.802-15-4]  
"Information technology - Telecommunications and  
information exchange between systems - Local and  
metropolitan area networks - Specific requirements - Part  
15.4: Wireless Medium Access Control (MAC) and Physical

Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)", IEEE Standard 802.15.4, September 2006, <<http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf>>.

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <[http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6256] Eddy, W. and E. Davies, "Using Self-Delimiting Numeric Values in Protocols", RFC 6256, May 2011.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, February 2012.
- [SOS12] Arkko, J. and H. Tschofenig, "Conclusions from the Workshop on Smart Object Security", Workshop on Smart Object Security, March 2012, <<http://www.lix.polytechnique.fr/hipercom/SmartObjectSecurity/slides/sos-conclusions.ppt>>.
- [USENIX01] Dean, D. and A. Stubblefield, "Using Client Puzzles to Protect TLS", 10th USENIX Security Symposium, August 2001, <[http://static.usenix.org/events/sec01/full\\_papers/dean/dean.pdf](http://static.usenix.org/events/sec01/full_papers/dean/dean.pdf)>.

## Appendix A. Templates

When elliptic curve cryptography is used, building and parsing the bodies of Certificate, ServerKeyExchange and ClientKeyExchange messages mainly involves the encoding and decoding of elliptic curve points. The points are encapsulated in a mix of DTLS structures and ASN.1 sequences. For a given elliptic curve, some parts of a message body are static, which allows using pre-composed messages instead of writing lots of memory consuming code pertaining to DTLS and ASN.1.

This appendix provides templates for the bodies of the Certificate, ServerKeyExchange and ClientKeyExchange messages used in a DTLS handshake with raw public key certificates [I-D.ietf-tls-oob-pubkey] and the ECDHE\_ECDSA key exchange [RFC4492].

The templates are given for the named curves secp256r1, secp384r1 and secp521r1; these curves are equivalent to the NIST P-256, P-384, and P-521 curves. They are required in [I-D.mcgrewe-tls-aes-ccm-ecc]. The same curve is used in each case for both the raw public key certificate and the ephemeral keys. Points are represented in uncompressed point format.

Note: The templates have not been independently verified yet.

## A.1. secp256r1

Raw Public Key Certificate:

```

30 59 30 13 06 07 2a 86  48 ce 3d 02 01 06 08 2a
86 48 ce 3d 03 01 07 03  42 00 04  _ _ _ _ _
_ _ _ _ _ _ _ _ _ _  _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _  _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _  _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _  _ _ _ _ _
```

ECDSA-capable public key (x, y)

## Server Key Exchange:

```

03 00 17 41 04  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ 00 46 30 44 02 20 _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ 02 20 _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _

```

ephemeral ECDH public key (x, y) and ECDSA signature (r, s)

## Client Key Exchange:

```

41 04  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _

```

ephemeral ECDH public key (x, y)

## A.2. secp384r1

## Raw Public Key Certificate:

```

30 76 30 10 06 07 2a 86 48 ce 3d 02 01 06 05 2b
81 04 00 22 03 62 00 04  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _
_ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _  _ _ _

```

ECDSA-capable public key (x, y)

## Server Key Exchange:

```

03 00 18 61 04  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ 00 66 30 64 02 30  _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 02 30  _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

```

ephemeral ECDH public key (x, y) and ECDSA signature (r, s)

## Client Key Exchange:

```

61 04  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _

```

ephemeral ECDH public key (x, y)

## A.3. secp521r1

## Raw Public Key Certificate:

```

30 81 9b 30 10 06 07 2a 86 48 ce 3d 02 01 06 05
2b 81 04 00 23 03 81 86 00 04  _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

```

ECDSA-capable public key (x, y)

## Server Key Exchange:

```

03 00 19 85 04  _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ 00 8b 30 81 88 02 42
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ 02 42 _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _

```

ephemeral ECDH public key (x, y) and ECDSA signature (r, s)

## Client Key Exchange:

```

85 04  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _

```

ephemeral ECDH public key (x, y)

Authors' Addresses

Klaus Hartke  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63905  
Email: hartke@tzi.org

Olaf Bergmann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63904  
Email: bergmann@tzi.org





Network Working Group  
Internet-Draft

Intended status: Informational  
Expires: January 12, 2013

X. He  
YC.Ma  
Hitachi (China) Research and  
Development Corporation  
Z.Cao  
China Mobile  
July 16, 2012

Energy Aware Proxy Discovery for CoAP  
draft-he-core-energy-aware-pd-01.txt

## Abstract

CoRE defines a mechanism for resource discovery based on Web linking with discovery, registration, modification, and other procedures. But energy efficiency is very important for resource constrained devices. This specification shows an efficient method for CoAP proxy finding the resource from end-points by reducing multicast messages.

The current version -01 of this document is just an initial draft that is modified according to related comments.

## 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 January 12, 2013.

## Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|  |   |
|--|---|
| 1. Introduction . . . . .                  | 3 |
| 2. Resource discovery analysis. . . . .    | 4 |
| 3. Energy aware proxy discovery . . . . .  | 5 |
| 3.1. Reduced Protocol Operations . . . . . | 5 |
| 3.2. Example . . . . .                     | 6 |
| 4. IANA Consideration . . . . .            | 7 |
| 5. Security Considerations . . . . .       | 7 |
| 6. Normative References . . . . .          | 7 |

## 1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes and networks. CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation [I-D.shelby-core-coap-req]. As being the main work of CoRE, CoAP defined a proxy mechanism for CoAP end-point, that proxy can be tasked by CoAP clients to perform requests on their behalf [I-D.ietf-core-coap].

Since in many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources [I-D.shelby-core-resource-directory].

The proxy mechanism should support this RD function in resource constrained environments. There are several methods to discovery proxy by a CoAP end-point, including assuming a default location (e.g. on an Edge Router in a LoWPAN), by assigning an anycast address to the proxy, using DHCP, or using the CoRE Link Format. While reducing energy consumption is essential for battery operated nodes in some devices, which is one of the most important work in M2M communication. Node's energy usage depends on network messages it has to receive and or respond. Thus the discovery procedure should be optimized with energy aware consideration.

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 [RFC2119].

## 2. Resource discovery analysis

As mentioned in other drafts, resource discovery is performed by sending either a multicast or unicast GET request to `/.well-known/core` and including a Resource Type (`rt`) parameter [I-D.ietf-core-link-format] with the value "core-rd" in the query string. Upon success, the response will contain a payload with a link format entry for each RD discovered, with the URL indicating the root resource of the RD.

After discovering the location of an RD, an end-point MAY register its resources to the RD's registration interface. This interface accepts a POST from an end-point containing the list of resources to be added to the directory as the message payload in the CoRE Link Format along with query string parameters indicating the name of the end-point, an optional node identifier and the lifetime of the registration. Upon success, the response will be 2.01 "Created" [I-D.shelby-core-resource-directory].

That means once resource discovery needs twice communication process between end-point and proxy. For energy saving point of view, this procedure should be optimized. Another draft indicates more efficient method. A CoAP server that wants to make itself discoverable sends a POST request to the default discovery URI of any Candidate CoAP Server Discovery Server [I-D.bormann-core-simple-server-discovery].

This draft shows more details about energy aware proxy discovery mechanism for CoAP.

### 3. Energy aware proxy discovery

#### 3.1. Reduced Protocol Operations

The Energy aware proxy reduces discovery and registration processes into one.

The end-point acting as a server will use server IP address, the CoAP default port[I-D.ietf-core-coap], and the absolute path `"/.well-known/core"`[I-D.ietf-core-link-format] to build its POST request. And this request will be send to its Neighbor by unicast (as using 6LOWPAN or IPv6) or to a multicast address.

The POST request is a link-format message, which indicates the service list that requesting server wants to make known to the proxy.

End-point resources in the proxy are kept active for the period also indicated by the lifetime parameter. But unless the end-point needs to refreshing the data with update message, the data will be kept in this lifetime. Then the data will be delete automatically as expired.

The discovery interface is specified as follows:

Interaction: EP -> Proxy

Path: `/.well-known/core`

Method: POST

Content-Type: `application/link-format`

Parameters:

Lifetime (lt): Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, the value will be a default lifetime, (e.g.3600seconds)

Host (h): The host identifier or name of the registering node. The maximum length of this parameter is 63 octets. This parameter is combined with the Instance parameter (if any) to form the end-point name. If not included, the proxy MUST generate a unique Host name on behalf of the node.

Instance (ins): The instance of the end-point on this host, if there are multiple. The maximum length of this parameter is 63 octets.

Type (rt): The semantic type of end-point. The maximum length of this parameter is 63 octets.

Success: 2.01 "Created". The Location header of the new resource entry for the end-point could be e.g. in the form `/rd-base/{end-point name}`

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

### 3.2 Example

The following example shows an end-point with the name "node1" POST temperature resource to a proxy using this interface.

| End-point   | Neighbor<br>(as a proxy) |
|---|--------------------------|
| <pre>--- POST /.well-known/core?rt=core-rd-----&gt;</pre> |                          |
| <pre>&lt;-- 2.01 Created Location: /rd/node1 -----</pre>  |                          |

Req: POST coap://[ff02::1]/.well-known/core?rt=core-rd

Payload:

`</sensors/temp>;ct=41;rt="TemperatureC";if="sensor",`

Res: 2.01 Created

Location: /rd/node1

The second example shows an end-point with the name "node2" POST video resource to a proxy using this interface, but proxy cannot support this application data.

| End-point   | Neighbor<br>(as a proxy) |
|---|--------------------------|
| <pre>--- POST /.well-known/core?rt=core-rd-----&gt;</pre> |                          |
| <pre>&lt;-- 4.00 Bad Request-----</pre>                   |                          |

Req: POST coap://[ff02::1]/.well-known/core?rt=core-rd

Payload:

`</sensors/temp>;ct=41;rt="video";if="sensor",`

Res: 4.00 Bad Request

#### 4. Security Considerations

TBD.

#### 5. IANA Considerations

This document does not require any IANA actions.

#### 6. Normative References

[I-D.shelby-core-coap-req]

Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, "CoAP Requirements and Features", draft-shelby-core-coap-req-01 (work in progress), April 2010.

[I-D.ietf-core-coap]

Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-06, May 2011.

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

Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-07, July 2011.

[I-D.bormann-core-simple-server-discovery]

Bormann, C., "CoRE Simple Server Discovery" draft-bormann-core-simple-server-discovery-00, June, 2011

[I-D.shelby-core-resource-directory]

Shelby, Z., Krco, S., "CoRE Resource Directory" draft-shelby-core-resource-directory-01, September, 2011

Authors' Addresses

Xuan He  
Yuanchen Ma  
Hitachi R&D China  
301 of North Tower C, Raycom  
@ Kexuyuan Nanlu, Haidian District  
Beijing 100190  
China

Email:  
xhe@hitachi.cn  
ycma@hitachi.cn

Zhen Cao  
China Mobile  
Unit2, 28 Xuanwumenxi Ave,Xuanwu District  
Beijing 100053  
China

Email: zehn.cao@gmail.com



CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 18, 2012

C. Bormann  
Universitaet Bremen TZI  
Z. Shelby, Ed.  
Sensinode  
February 15, 2012

Blockwise transfers in CoAP  
draft-ietf-core-block-08

Abstract

CoAP is a RESTful transfer protocol for constrained nodes and networks. Basic CoAP messages work well for the small payloads we expect from temperature sensors, light switches, and similar building-automation devices. Occasionally, however, applications will need to transfer larger payloads -- for instance, for firmware updates. With HTTP, TCP does the grunt work of slicing large payloads up into multiple packets and ensuring that they all arrive and are handled in the right order.

CoAP is based on datagram transports such as UDP or DTLS, which limits the maximum size of resource representations that can be transferred without too much fragmentation. Although UDP supports larger payloads through IP fragmentation, it is limited to 64 KiB and, more importantly, doesn't really work well for constrained applications and networks.

Instead of relying on IP fragmentation, this specification extends basic CoAP with a pair of "Block" options, for transferring multiple blocks of information from a resource representation in multiple request-response pairs. In many important cases, the Block options enable a server to be truly stateless: the server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers.

In summary, the Block options provide a minimal way to transfer larger representations in a block-wise fashion.

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 August 18, 2012.

#### Copyright Notice

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

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

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .                             | 4  |
| 2. Block-wise transfers . . . . .                     | 6  |
| 2.1. The Block Options . . . . .                      | 6  |
| 2.2. Using the Block Options . . . . .                | 10 |
| 3. Examples . . . . .                                 | 13 |
| 4. The Size Option . . . . .                          | 20 |
| 5. HTTP Mapping Considerations . . . . .              | 22 |
| 6. IANA Considerations . . . . .                      | 24 |
| 7. Security Considerations . . . . .                  | 25 |
| 7.1. Mitigating Resource Exhaustion Attacks . . . . . | 25 |
| 7.2. Mitigating Amplification Attacks . . . . .       | 26 |
| 8. Acknowledgements . . . . .                         | 27 |
| 9. References . . . . .                               | 28 |
| 9.1. Normative References . . . . .                   | 28 |
| 9.2. Informative References . . . . .                 | 28 |
| Appendix A. Historical Note . . . . .                 | 29 |
| Authors' Addresses . . . . .                          | 30 |

## 1. Introduction

The CoRE WG is tasked with standardizing an Application Protocol for Constrained Networks/Nodes, CoAP. This protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC2616], while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

This objective requires restraint in a number of sometimes conflicting ways:

- o reducing implementation complexity in order to minimize code size,
- o reducing message sizes in order to minimize the number of fragments needed for each message (in turn to maximize the probability of delivery of the message), the amount of transmission power needed and the loading of the limited-bandwidth channel,
- o reducing requirements on the environment such as stable storage, good sources of randomness or user interaction capabilities.

CoAP is based on datagram transports such as UDP, which limit the maximum size of resource representations that can be transferred without creating unreasonable levels of IP fragmentation. In addition, not all resource representations will fit into a single link layer packet of a constrained network, which may cause adaptation layer fragmentation even if IP layer fragmentation is not required. Using fragmentation (either at the adaptation layer or at the IP layer) to enable the transport of larger representations is possible up to the maximum size of the underlying datagram protocol (such as UDP), but the fragmentation/reassembly process burdens the lower layers with conversation state that is better managed in the application layer.

This specification defines a pair of CoAP options to enable `_block-wise_` access to resource representations. The Block options provide a minimal way to transfer larger resource representations in a block-wise fashion. The overriding objective is to avoid creating conversation state at the server for block-wise GET requests. (It is impossible to fully avoid creating conversation state for POST/PUT, if the creation/replacement of resources is to be atomic; where that property is not needed, there is no need to create server conversation state in this case, either.)

In summary, this specification adds a pair of Block options to CoAP that can be used for block-wise transfers. Benefits of using these

options include:

- o Transfers larger than can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- o No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- o The transfer of each block is acknowledged, enabling retransmission if required.
- o Both sides have a say in the block size that actually will be used.
- o The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- o If needed, the Block options can also be used as is to provide random access to power-of-two sized blocks within a resource representation.

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, BCP 14 [RFC2119] and indicate requirement levels for compliant CoAP implementations.

In this document, the term "byte" is used in its now customary sense as a synonym for "octet".

Where bit arithmetic is explained, this document uses the notation familiar from the programming language C, except that the operator "\*\*\*" stands for exponentiation.

## 2. Block-wise transfers

As discussed in the introduction, there are good reasons to limit the size of datagrams in constrained networks:

- o by the maximum datagram size (~ 64 KiB for UDP)
- o by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- o by the desire to avoid adaptation layer fragmentation (60-80 bytes for 6LoWPAN [RFC4919])

When a resource representation is larger than can be comfortably transferred in the payload of a single CoAP datagram, a Block option can be used to indicate a block-wise transfer. As payloads can be sent both with requests and with responses, this specification provides two separate options for each direction of payload transfer.

In the following, the term "payload" will be used for the actual content of a single CoAP message, i.e. a single block being transferred, while the term "body" will be used for the entire resource representation that is being transferred in a block-wise fashion.

In most cases, all blocks being transferred for a body will be of the same size. The block size is not fixed by the protocol. To keep the implementation as simple as possible, the Block options support only a small range of power-of-two block sizes, from  $2^4$  (16) to  $2^{10}$  (1024) bytes. As bodies often will not evenly divide into the power-of-two block size chosen, the size need not be reached in the final block (but even for the final block, the chosen power-of-two size will still be indicated in the block size field of the Block option).

### 2.1. The Block Options

| Type | C/E      | Name   | Format | Length | Default       |
|------|----------|--------|--------|--------|---------------|
| 19   | Critical | Block1 | uint   | 1-3 B  | 0 (see below) |
| 17   | Critical | Block2 | uint   | 1-3 B  | 0 (see below) |

Table 1: Block Option Numbers

Both Block1 and Block2 options can be present both in request and response messages. In either case, the Block1 Option pertains to the request payload, and the Block2 Option pertains to the response

payload.

Hence, for the methods defined in [I-D.ietf-core-coap], Block1 is useful with the payload-bearing POST and PUT requests and their responses. Block2 is useful with GET, POST, and PUT requests and their payload-bearing responses (2.01, 2.02, 2.04, 2.05 -- see section "Payload" of [I-D.ietf-core-coap]).

(As a memory aid: Block\_1\_ pertains to the payload of the \_1st\_ part of the request-response exchange, i.e. the request, and Block\_2\_ pertains to the payload of the \_2nd\_ part of the request-response exchange, i.e. the response.)

Where Block1 is present in a request or Block2 in a response (i.e., in that message to the payload of which it pertains) it indicates a block-wise transfer and describes how this block-wise payload forms part of the entire body being transferred ("descriptive usage"). Where it is present in the opposite direction, it provides additional control on how that payload will be formed or was processed ("control usage").

Implementation of either Block option is intended to be optional. However, when it is present in a CoAP message, it MUST be processed (or the message rejected); therefore it is identified as a critical option. It MUST NOT occur more than once.

Three items of information may need to be transferred in a Block option:

- o The size of the block (SZX);
- o whether more blocks are following (M);
- o the relative number of the block (NUM) within a sequence of blocks with the given size.

The value of the option is a 1-, 2- or 3-byte integer which encodes these three fields, see Figure 1.

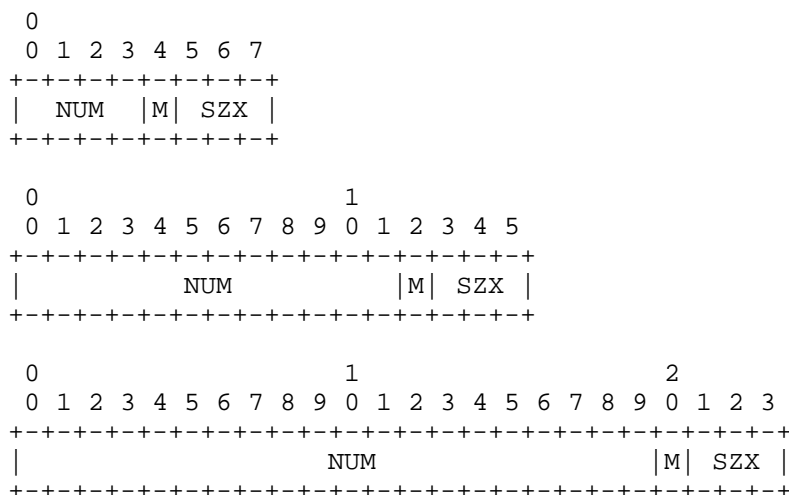


Figure 1: Block option value

The block size is encoded as a three-bit unsigned integer (0 for  $2^{*}4$  to 6 for  $2^{*}10$  bytes), which we call the "SZX" (size exponent); the actual block size is then  $2^{*}(SZX + 4)$ . SZX is transferred in the three least significant bits of the option value (i.e., "val & 7" where "val" is the value of the option).

The fourth least significant bit, the M or "more" bit ("val & 8"), indicates whether more blocks are following or the current block-wise transfer is the last block being transferred.

The option value divided by sixteen (the NUM field) is the sequence number of the block currently being transferred, starting from zero. The current transfer is therefore about the "size" bytes starting at byte  $NUM \ll (SZX + 4)$ . (Note that, as an implementation convenience,  $(val \ll \sim 0xF) \ll (val \ll 7)$ , i.e. the option value with the last 4 bits masked out, shifted to the left by the value of SZX, gives the byte position of the block.)

The default value of both the Block1 and the Block2 Option is zero, indicating that the current block is the first and only block of the transfer (block number 0, M bit not set); however, there is no explicit size implied by this default value.

More specifically, within the option value of a Block1 or Block2 Option, the meaning of the option fields is defined as follows:



NUM: Block Number. The block number is a variable-size (4, 12, or 20 bit) unsigned integer (uint, see Appendix A of [I-D.ietf-core-coap]) indicating the block number being requested or provided. Block number 0 indicates the first block of a body.

M: More Flag (not last block). For descriptive usage, this flag, if unset, indicates that the payload in this message is the last block in the body; when set it indicates that there are one or more additional blocks available. When a Block2 Option is used in a request to retrieve a specific block number ("control usage"), the M bit MUST be sent as zero and ignored on reception. (In a Block1 Option in a response, the M flag is used to indicate atomicity, see below.)

SZX: Block Size. The block size is a three-bit unsigned integer indicating the size of a block to the power of two. Thus block size =  $2^{(SZX + 4)}$ . The allowed values of SZX are 0 to 6, i.e., the minimum block size is  $2^{(0+4)} = 16$  and the maximum is  $2^{(6+4)} = 1024$ . The value 7 for SZX (which would indicate a block size of 2048) is reserved, i.e. MUST NOT be sent and MUST lead to a 4.00 Bad Request response code upon reception in a request.

The Block options are used in one of three roles:

- o In descriptive usage, i.e. a Block2 Option in a response (e.g., a 2.05 response for GET), or a Block1 Option in a request (e.g., PUT or POST):
  - \* The NUM field in the option value describes what block number is contained in the payload of this message.
  - \* The M bit indicates whether further blocks are required to complete the transfer of that body.
  - \* The block size given by SZX MUST match the size of the payload in bytes, if the M bit is set. (SZX does not govern the payload size if M is unset). For Block2, if the request suggested a larger value of SZX, the next request MUST move SZX down to the size given here. (The effect is that, if the server uses the smaller of its preferred block size and the one requested, all blocks for a body use the same block size.)
- o A Block2 Option in control usage in a request (e.g., GET):
  - \* The NUM field in the Block2 Option gives the block number of the payload that is being requested to be returned in the response.

- \* In this case, the M bit has no function and MUST be set to zero.
- \* The block size given (SZX) suggests a block size (in the case of block number 0) or repeats the block size of previous blocks received (in the case of block numbers other than 0).
- o A Block1 Option in control usage in a response (e.g., a 2.xx response for a PUT or POST request):
  - \* The NUM field of the Block1 Option indicates what block number is being acknowledged.
  - \* If the M bit was set in the request, the server can choose whether to act on each block separately, with no memory, or whether to handle the request for the entire body atomically, or any mix of the two. If the M bit is also set in the response, it indicates that this response does not carry the final response code to the request, i.e. the server collects further blocks and plans to implement the request atomically (e.g., acts only upon reception of the last block of payload). Conversely, if the M bit is unset even though it was set in the request, it indicates the block-wise request was enacted now specifically for this block, and the response carries the final response to this request (and to any previous ones with the M bit set in the response's Block1 Option in this sequence of block-wise transfers); the client is still expected to continue sending further blocks, the request method for which may or may not also be enacted per-block.
  - \* Finally, the SZX block size given in a control Block1 Option indicates the largest block size preferred by the server for transfers toward the resource that is the same or smaller than the one used in the initial exchange; the client SHOULD use this block size or a smaller one in all further requests in the transfer sequence, even if that means changing the block size (and possibly scaling the block number accordingly) from now on.

## 2.2. Using the Block Options

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. As specified in [I-D.ietf-core-coap], each of these message exchanges uses their own CoAP Message ID.

When a request is answered with a response carrying a Block2 Option with the M bit set, the requester may retrieve additional blocks of

the resource representation by sending further requests with the same options and a Block2 Option giving the block number and block size desired. In a request, the client **MUST** set the M bit of a Block2 Option to zero and the server **MUST** ignore it on reception.

To influence the block size used in a response, the requester also uses the Block2 Option, giving the desired size, a block number of zero and an M bit of zero. A server **MUST** use the block size indicated or a smaller size. Any further block-wise requests for blocks beyond the first one **MUST** indicate the same block size that was used by the server in the response for the first request that gave a desired size using a Block2 Option.

Once the Block2 Option is used by the requester, all requests in a single block-wise transfer **MUST** ultimately use the same size, except that there may not be enough content to fill the last block (the one returned with the M bit not set). (Note that the client may start using the Block2 Option in a second request after a first request without a Block2 Option resulted in a Block option in the response.) The server **SHOULD** use the block size indicated in the request option or a smaller size, but the requester **MUST** take note of the actual block size used in the response it receives to its initial request and proceed to use it in subsequent requests. The server behavior **MUST** ensure that this client behavior results in the same block size for all responses in a sequence (except for the last one with the M bit not set, and possibly the first one if the initial request did not contain a Block2 Option).

Block-wise transfers can be used to GET resources the representations of which are entirely static (not changing over time at all, such as in a schema describing a device), or for dynamically changing resources. In the latter case, the Block2 Option **SHOULD** be used in conjunction with the ETag Option, to ensure that the blocks being reassembled are from the same version of the representation: The server **SHOULD** include an ETag option in each response. If an ETag option is available, the client's reassembler, when reassembling the representation from the blocks being exchanged, **MUST** compare ETag Options. If the ETag Options do not match in a GET transfer, the requester has the option of attempting to retrieve fresh values for the blocks it retrieved first. To minimize the resulting inefficiency, the server **MAY** cache the current value of a representation for an ongoing sequence of requests. The client **MAY** facilitate identifying the sequence by using the Token Option with a non-default value. Note well that this specification makes no requirement for the server to establish any state; however, servers that offer quickly changing resources may thereby make it impossible for a client to ever retrieve a consistent set of blocks.

In a request with a request payload (e.g., PUT or POST), the Block1 Option refers to the payload in the request (descriptive usage).

In response to a request with a payload (e.g., a PUT or POST transfer), the block size given in the Block1 Option indicates the block size preference of the server for this resource (control usage). Obviously, at this point the first block has already been transferred by the client without benefit of this knowledge. Still, the client SHOULD heed the preference and, for all further blocks, use the block size preferred by the server or a smaller one. Note that any reduction in the block size may mean that the second request starts with a block number larger than one, as the first request already transferred multiple blocks as counted in the smaller size.

To counter the effects of adaptation layer fragmentation on packet delivery probability, a client may want to give up retransmitting a request with a relatively large payload even before MAX\_RETRANSMIT has been reached, and try restating the request as a block-wise transfer with a smaller payload. Note that this new attempt is then a new message-layer transaction and requires a new Message ID. (Because of the uncertainty whether the request or the acknowledgement was lost, this strategy is useful mostly for idempotent requests.)

In a blockwise transfer of a request payload (e.g., a PUT or POST) that is intended to be implemented in an atomic fashion at the server, the actual creation/replacement takes place at the time the final block, i.e. a block with the M bit unset in the Block1 Option, is received. If not all previous blocks are available at the server at this time, the transfer fails and error code 4.08 (Request Entity Incomplete) MUST be returned. The error code 4.13 (Request Entity Too Large) can be returned at any time by a server that does not currently have the resources to store blocks for a block-wise request payload transfer that it would intend to implement in an atomic fashion. (Note that a 4.13 response to a request that does not employ Block1 is a hint for the client to try sending Block1, and a 4.13 response with a smaller SZX in the Block1 than requested is a hint to try a smaller SZX.)

If multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations are possible, the requester SHOULD use the Token Option to clearly separate the different sequences. In this case, when reassembling the representation from the blocks being exchanged to enable atomic processing, the reassembler MUST compare any Token Options present (and, as usual, taking an absent Token Option to default to the empty Token). If atomic processing is not desired, there is no need to process the Token Option (but it is still returned in the response as usual).

### 3. Examples

This section gives a number of short examples with message flows for a block-wise GET, and for a PUT or POST. These examples demonstrate the basic operation, the operation in the presence of retransmissions, and examples for the operation of the block size negotiation.

In all these examples, a Block option is shown in a decomposed way separating the kind of Block option (1 or 2), block number (NUM), more bit (M), and block size exponent ( $2^{*(SZX+4)}$ ) by slashes. E.g., a Block2 Option value of 33 would be shown as 2/2/0/32), or a Block1 Option value of 59 would be shown as 1/3/1/128.

The first example (Figure 2) shows a GET request that is split into three blocks. The server proposes a block size of 128, and the client agrees. The first two ACKs contain 128 bytes of payload each, and third ACK contains between 1 and 128 bytes.

| CLIENT   | SERVER |
|--|--------|
| CON [MID=1234], GET, /status                   | -----> |
| <----- ACK [MID=1234], 2.05 Content, 2/0/1/128 |        |
| CON [MID=1235], GET, /status, 2/1/0/128        | -----> |
| <----- ACK [MID=1235], 2.05 Content, 2/1/1/128 |        |
| CON [MID=1236], GET, /status, 2/2/0/128        | -----> |
| <----- ACK [MID=1236], 2.05 Content, 2/2/0/128 |        |

Figure 2: Simple blockwise GET

In the second example (Figure 3), the client anticipates the blockwise transfer (e.g., because of a size indication in the link-format description [I-D.ietf-core-link-format]) and sends a size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

| CLIENT  |        | SERVER |
|---|--------|--------|
| CON [MID=1234], GET, /status, 2/0/0/64        | -----> |        |
| <----- ACK [MID=1234], 2.05 Content, 2/0/1/64 |        |        |
| CON [MID=1235], GET, /status, 2/1/0/64        | -----> |        |
| <----- ACK [MID=1235], 2.05 Content, 2/1/1/64 |        |        |
| :   |        | :      |
| :   | ...    | :      |
| :   |        | :      |
| CON [MID=1238], GET, /status, 2/4/0/64        | -----> |        |
| <----- ACK [MID=1238], 2.05 Content, 2/4/1/64 |        |        |
| CON [MID=1239], GET, /status, 2/5/0/64        | -----> |        |
| <----- ACK [MID=1239], 2.05 Content, 2/5/0/64 |        |        |

Figure 3: Blockwise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a blockwise transfer, and unhappy with the size chosen unilaterally by the server. As it did not send a size proposal initially, the negotiation only influences the size from the second message exchange onward. Since the client already obtained both the first and second 64-byte block in the first 128-byte exchange, it goes on requesting the third 64-byte block ("2/0/64"). None of this is (or needs to be) understood by the server, which simply responds to the requests as it best can.

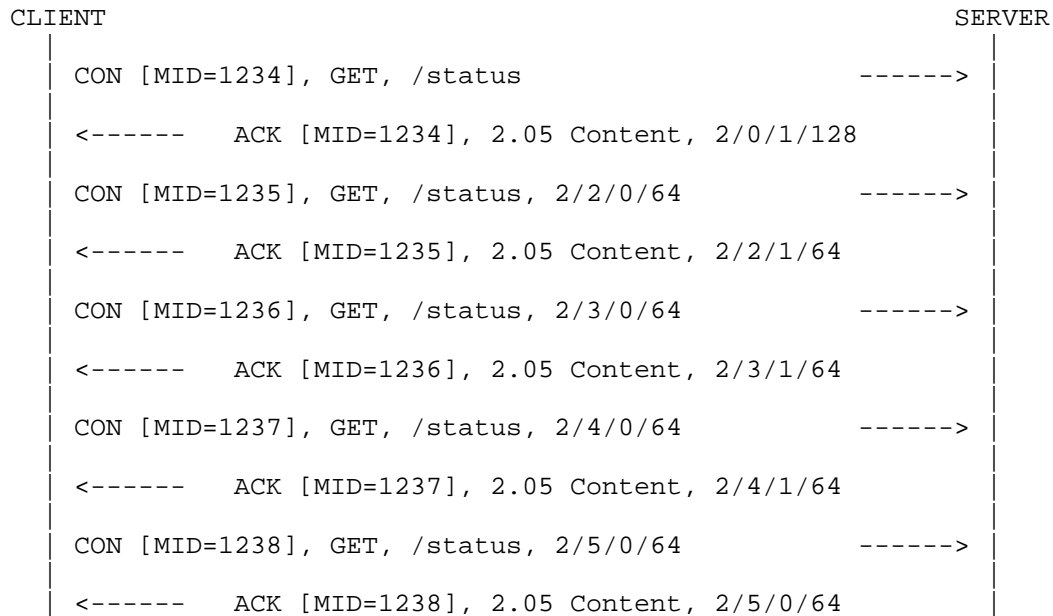


Figure 4: Blockwise GET with late negotiation

In all these (and the following) cases, retransmissions are handled by the CoAP message exchange layer, so they don't influence the block operations (Figure 5, Figure 6).

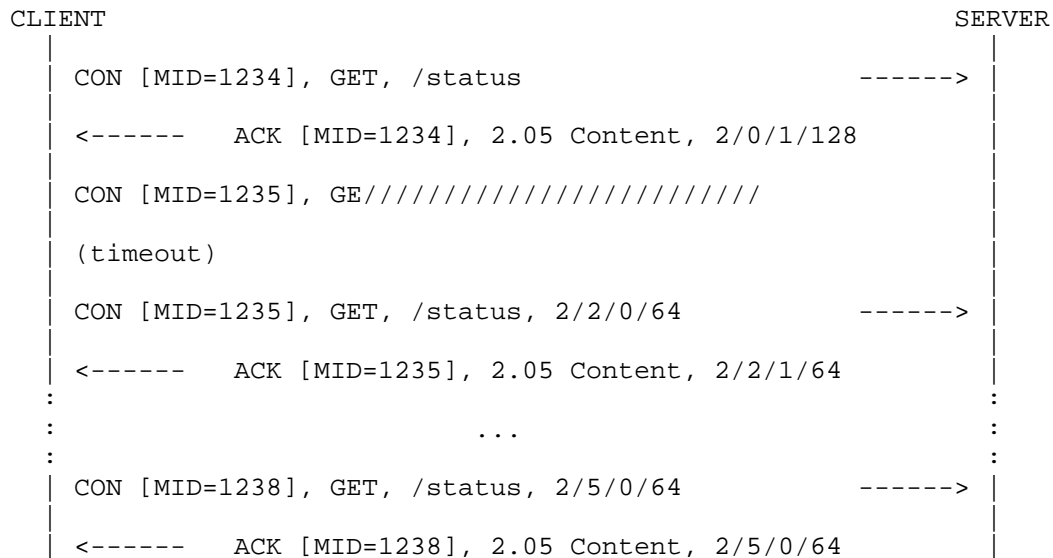


Figure 5: Blockwise GET with late negotiation and lost CON

| CLIENT   |        | SERVER |
|--|--------|--------|
| CON [MID=1234], GET, /status                             | -----> |        |
| <----- ACK [MID=1234], 2.05 Content, 2/0/1/128           |        |        |
| CON [MID=1235], GET, /status, 2/2/0/64                   | -----> |        |
| //tent, 2/2/1/64 |        |        |
| (timeout)  |        |        |
| CON [MID=1235], GET, /status, 2/2/0/64                   | -----> |        |
| <----- ACK [MID=1235], 2.05 Content, 2/2/1/64            |        |        |
| :  |        | :      |
| :  | ...    | :      |
| :  |        | :      |
| CON [MID=1238], GET, /status, 2/5/0/64                   | -----> |        |
| <----- ACK [MID=1238], 2.05 Content, 2/5/0/64            |        |        |

Figure 6: Blockwise GET with late negotiation and lost ACK

The following examples demonstrate a PUT exchange; a POST exchange looks the same, with different requirements on atomicity/idempotence. To ensure that the blocks relate to the same version of the resource representation carried in the request, the client in Figure 7 sets the Token to "v17" in all requests. Note that, similar to GET, the responses to the requests that have a more bit in the request Block1 Option are provisional; only the final response tells the client that the PUT succeeded.



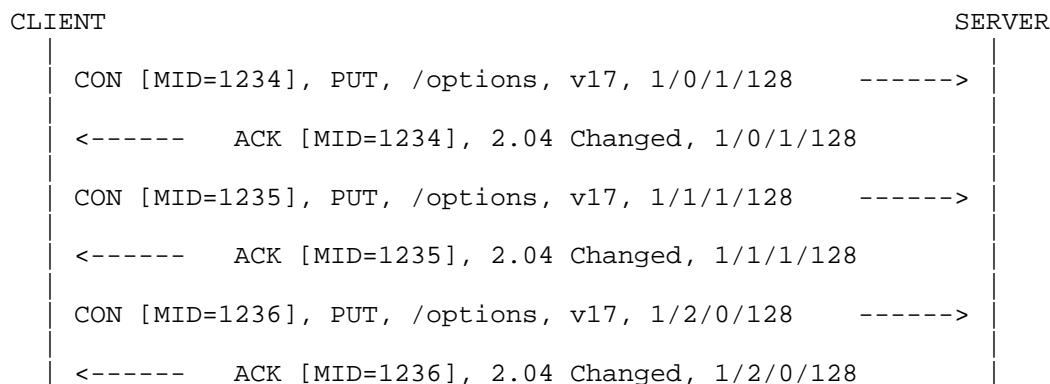


Figure 7: Simple atomic blockwise PUT

A stateless server that simply builds/updates the resource in place (statelessly) may indicate this by not setting the more bit in the response (Figure 8); in this case, the response codes are valid separately for each block being updated. This is of course only an acceptable behavior of the server if the potential inconsistency present during the run of the message exchange sequence does not lead to problems, e.g. because the resource being created or changed is not yet or not currently in use.

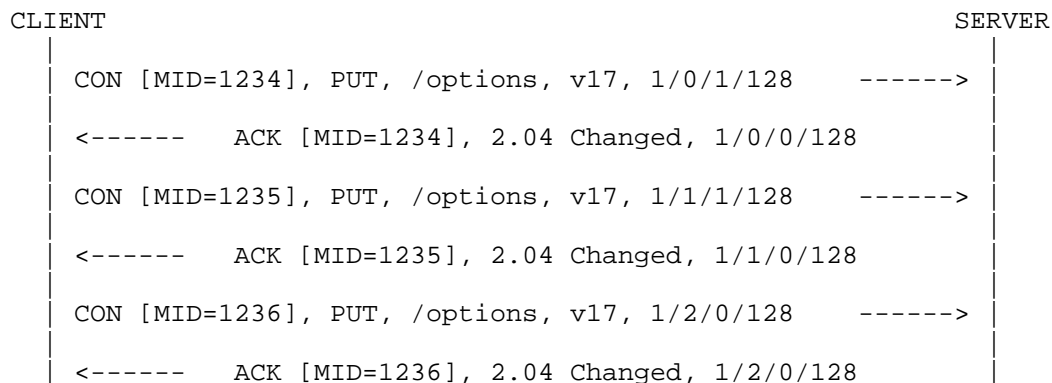


Figure 8: Simple stateless blockwise PUT

Finally, a server receiving a blockwise PUT or POST may want to indicate a smaller block size preference (Figure 9). In this case, the client SHOULD continue with a smaller block size; if it does, it MUST adjust the block number to properly count in that smaller size.

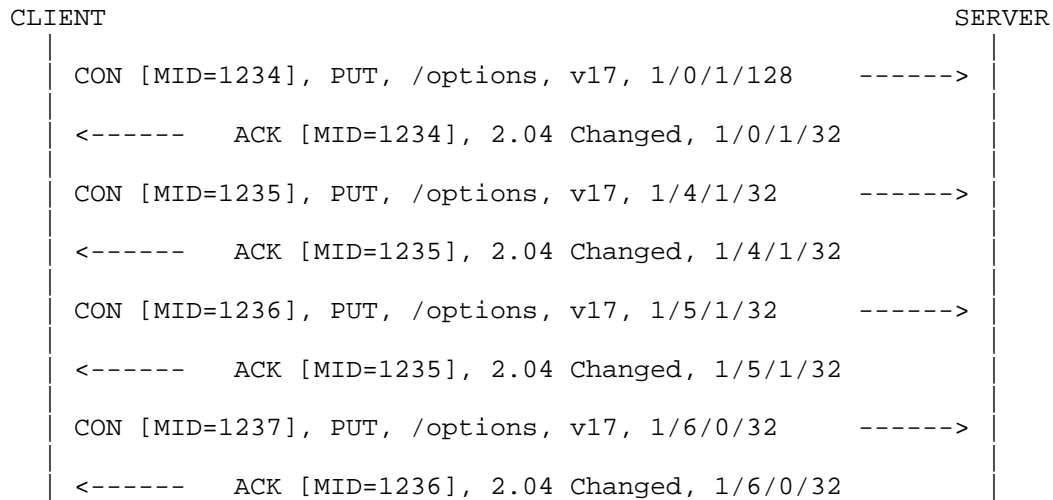


Figure 9: Simple atomic blockwise PUT with negotiation

Block options may be used in both directions of a single exchange. The following example demonstrates a blockwise POST request, resulting in a separate blockwise response. The client in Figure 10 sets the Token to "37a" in all requests, which is echoed in all response CONs in the separate response.

| CLIENT  | SERVER |
|---|--------|
| CON [MID=1234], POST, /soap, 37a, 1/0/1/128         | -----> |
| <----- ACK [MID=1234], 2.01 Created, 1/0/1/128      |        |
| CON [MID=1235], POST, /soap, 37a, 1/1/1/128         | -----> |
| <----- ACK [MID=1235], 2.01 Created, 1/1/1/128      |        |
| CON [MID=1236], POST, /soap, 37a, 1/2/0/128         | -----> |
| <----- ACK [MID=1236], 0, 1/2/0/128                 |        |
| <----- CON [MID=4712], 2.01 Created, 37a, 2/0/1/128 |        |
| ACK [MID=4712], 0, 2/0/1/128                        | -----> |
| <----- CON [MID=4713], 2.01 Created, 37a, 2/1/1/128 |        |
| ACK [MID=4713], 0, 2/1/1/128                        | -----> |
| <----- CON [MID=4714], 2.01 Created, 37a, 2/2/1/128 |        |
| ACK [MID=4714], 0, 2/2/1/128                        | -----> |
| <----- CON [MID=4715], 2.01 Created, 37a, 2/3/0/128 |        |
| ACK [MID=4715], 0, 2/3/0/128                        | -----> |

Figure 10: Atomic blockwise POST with separate blockwise response

#### 4. The Size Option

In many cases when transferring a large resource representation block by block, it is advantageous to know the total size early in the process. Some indication may be available from the maximum size estimate attribute "sz" provided in a resource description [I-D.ietf-core-link-format]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

The Size Option may be used for three purposes:

- o in a request, to ask the server to provide a size estimate in the response ("size request"). For this usage, the value MUST be set to 0.
- o in a response carrying a Block2 Option, to indicate the current estimate the server has of the total size of the resource representation.
- o in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation.

A size request can be easily distinguished from a size indication, as the third case is not useful for a GET or DELETE, and an actual size indication of 0 would either be overridden by the actual size of the payload for a PUT or POST or would not be useful.

In the latter two cases ("size indication"), the value of the option is the current estimate, measured in bytes.

The Size Option is "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Option MUST NOT occur more than once.

| Type | C/E      | Name | Format | Length | Default |
|------|----------|------|--------|--------|---------|
| 18   | Elective | Size | uint   | 0-4 B  | (none)  |

#### Implementation Notes:

- o As a quality of implementation consideration, blockwise transfers for which the total size considerably exceeds the size of one block are expected to include size indications, whenever those can be provided without undue effort (preferably with the first block exchanged). If the size estimate does not change, the indication

does not need to be repeated for every block.

- o The end of a blockwise transfer is governed by the M bits in the Block Options, not by exhausting the size estimates exchanged.
- o As usual for an option of type uint, the value 0 is best expressed as an empty option (0 bytes). There is no default value.

## 5. HTTP Mapping Considerations

In this subsection, we give some brief examples for the influence the Block options might have on intermediaries that map between CoAP and HTTP.

For mapping CoAP requests to HTTP, the intermediary may want to map the sequence of block-wise transfers into a single HTTP transfer. E.g., for a GET request, the intermediary could perform the HTTP request once the first block has been requested and could then fulfill all further block requests out of its cache. A constrained implementation may not be able to cache the entire object and may use a combination of TCP flow control and (in particular if timeouts occur) HTTP range requests to obtain the information necessary for the next block transfer at the right time.

For PUT or POST requests, there is more variation in how HTTP servers might implement ranges. Some WebDAV servers do, but in general the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer within one HTTP request. If enough buffering is available, this request can be started when the last CoAP block is received. A constrained implementation may want to relieve its buffering by already starting to send the HTTP request at the time the first CoAP block is received; any HTTP 408 status code that indicates that the HTTP server became impatient with the resulting transfer can then be mapped into a CoAP 4.08 response code (similarly, 413 maps to 4.13).

For mapping HTTP to CoAP, the intermediary may want to map a single HTTP transfer into a sequence of block-wise transfers. If the HTTP client is too slow delivering a request body on a PUT or POST, the CoAP server might time out and return a 4.08 response code, which in turn maps well to an HTTP 408 status code (again, 4.13 maps to 413). HTTP range requests received on the HTTP side may be served out of a cache and/or mapped to GET requests that request a sequence of blocks overlapping the range.

(Note that, while the semantics of CoAP 4.08 and HTTP 408 differ, this difference is largely due to the different way the two protocols are mapped to transport. HTTP has an underlying TCP connection, which supplies connection state, so a HTTP 408 status code can immediately be used to indicate that a timeout occurred during transmitting a request through that active TCP connection. The CoAP 4.08 response code indicates one or more missing blocks, which may be due to timeouts or resource constraints; as there is no connection state, there is no way to deliver such a response immediately; instead, it is delivered on the next block transfer. Still, HTTP 408 is probably the best mapping back to HTTP, as the timeout is the most

likely cause for a CoAP 4.08. Note that there is no way to distinguish a timeout from a missing block for a server without creating additional state, the need for which we want to avoid.)

## 6. IANA Considerations

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

| Number | Name   | Reference |
|--------|--------|-----------|
| 17     | Block2 | [RFCXXXX] |
| 18     | Size   | [RFCXXXX] |
| 19     | Block1 | [RFCXXXX] |

Table 2: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [I-D.ietf-core-coap]:

| Code | Description                    | Reference |
|------|--------------------------------|-----------|
| 136  | 4.08 Request Entity Incomplete | [RFCXXXX] |

Table 3: CoAP Response Codes



## 7. Security Considerations

Providing access to blocks within a resource may lead to surprising vulnerabilities. Where requests are not implemented atomically, an attacker may be able to exploit a race condition or confuse a server by inducing it to use a partially updated resource representation. Partial transfers may also make certain problematic data invisible to intrusion detection systems; it is RECOMMENDED that an intrusion detection system (IDS) that analyzes resource representations transferred by CoAP implement the Block options to gain access to entire resource representations. Still, approaches such as transferring even-numbered blocks on one path and odd-numbered blocks on another path, or even transferring blocks multiple times with different content and obtaining a different interpretation of temporal order at the IDS than at the server, may prevent an IDS from seeing the whole picture. These kinds of attacks are well understood from IP fragmentation and TCP segmentation; CoAP does not add fundamentally new considerations.

Where access to a resource is only granted to clients making use of a specific security association, all blocks of that resource MUST be subject to the same security checks; it MUST NOT be possible for unprotected exchanges to influence blocks of an otherwise protected resource. As a related consideration, where object security is employed, PUT/POST should be implemented in the atomic fashion, unless the object security operation is performed on each access and the creation of unusable resources can be tolerated.

Misleading size indications may be used by an attacker to induce buffer overflows in poor implementations, for which the usual considerations apply.

### 7.1. Mitigating Resource Exhaustion Attacks

Certain blockwise requests may induce the server to create state, e.g. to create a snapshot for the blockwise GET of a fast-changing resource to enable consistent access to the same version of a resource for all blocks, or to create temporary resource representations that are collected until pressed into service by a final PUT or POST with the more bit unset. All mechanisms that induce a server to create state that cannot simply be cleaned up create opportunities for denial-of-service attacks. Servers SHOULD avoid being subject to resource exhaustion based on state created by untrusted sources. But even if this is done, the mitigation may cause a denial-of-service to a legitimate request when it is drowned out by other state-creating requests. Wherever possible, servers should therefore minimize the opportunities to create state for untrusted sources, e.g. by using stateless approaches.

Performing segmentation at the application layer is almost always better in this respect than at the transport layer or lower (IP fragmentation, adaptation layer fragmentation), e.g. because there is application layer semantics that can be used for mitigation or because lower layers provide security associations that can prevent attacks. However, it is less common to apply timeouts and keepalive mechanisms at the application layer than at lower layers. Servers MAY want to clean up accumulated state by timing it out (cf. response code 4.08), and clients SHOULD be prepared to run blockwise transfers in an expedient way to minimize the likelihood of running into such a timeout.

## 7.2. Mitigating Amplification Attacks

[I-D.ietf-core-coap] discusses the susceptibility of CoAP end-points for use in amplification attacks.

A CoAP server can reduce the amount of amplification it provides to an attacker by offering large resource representations only in relatively small blocks. With this, e.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

## 8. Acknowledgements

Much of the content of this draft is the result of discussions with the [I-D.ietf-core-coap] authors, and via many CoRE WG discussions. Tokens were suggested by Gilman Tolle and refined by Klaus Hartke.

Charles Palmer provided extensive editorial comments to a previous version of this draft, some of which the authors hope to have covered in this version. Esko Dijk reviewed a more recent version, leading to a number of further editorial improvements.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft.

## 9. References

### 9.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-08 (work in progress), October 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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, June 1999.

### 9.2. Informative References

- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format",  
draft-ietf-core-link-format-11 (work in progress),  
January 2012.
- [REST] Fielding, R., "Architectural Styles and the Design of  
Network-based Software Architectures", 2000.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6  
over Low-Power Wireless Personal Area Networks (6LoWPANs):  
Overview, Assumptions, Problem Statement, and Goals",  
RFC 4919, August 2007.

## Appendix A. Historical Note

(This appendix to be deleted by the RFC editor.)

An earlier version of this draft used a single option:

| Type | C/E      | Name  | Format | Length | Default       |
|------|----------|-------|--------|--------|---------------|
| 13   | Critical | Block | uint   | 1-3 B  | 0 (see below) |

Note that this option number has since been reallocated in [I-D.ietf-core-coap]; no backwards compatibility is provided after July 1st, 2011.

Authors' Addresses

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Fax: +49-421-218-7000  
Email: cabo@tzi.org

Zach Shelby (editor)  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
Finland

Phone: +358407796297  
Email: zach@sensinode.com



CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 17, 2013

Z. Shelby  
Sensinode  
K. Hartke  
C. Bormann  
Universitaet Bremen TZI  
B. Frank  
SkyFoundry  
July 16, 2012

Constrained Application Protocol (CoAP)  
draft-ietf-core-coap-11

Abstract

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained (e.g., low-power, lossy) networks. The nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while constrained networks such as 6LoWPAN often have high packet error rates and a typical throughput of 10s of kbit/s. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.

CoAP provides a request/response interaction model between application end-points, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP easily interfaces with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

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 January 17, 2013.



## Copyright Notice

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

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

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .   | 6  |
| 1.1. Features . . . . .   | 6  |
| 1.2. Terminology . . . . .  | 7  |
| 2. Constrained Application Protocol . . . . .                     | 9  |
| 2.1. Messaging Model . . . . .                                    | 10 |
| 2.2. Request/Response Model . . . . .                             | 11 |
| 2.3. Intermediaries and Caching . . . . .                         | 14 |
| 2.4. Resource Discovery . . . . .                                 | 14 |
| 3. Message Format . . . . .                                       | 14 |
| 3.1. Header Format . . . . .                                      | 15 |
| 3.2. Option Format . . . . .                                      | 16 |
| 3.3. Option Value Formats . . . . .                               | 17 |
| 3.3.1. uint . . . . .   | 17 |
| 3.3.2. string . . . . .   | 18 |
| 3.3.3. opaque . . . . .   | 18 |
| 3.3.4. empty . . . . .  | 18 |
| 4. Message Transmission . . . . .                                 | 18 |
| 4.1. Messages and Endpoints . . . . .                             | 19 |
| 4.2. Messages Transmitted Reliably . . . . .                      | 19 |
| 4.3. Messages Transmitted Without Reliability . . . . .           | 20 |
| 4.4. Message Correlation . . . . .                                | 21 |
| 4.5. Message Deduplication . . . . .                              | 21 |
| 4.6. Message Size . . . . .                                       | 22 |
| 4.7. Congestion Control . . . . .                                 | 23 |
| 4.8. Transmission Parameters . . . . .                            | 24 |
| 4.8.1. Changing The Parameters . . . . .                          | 24 |
| 4.8.2. Time Values derived from Transmission Parameters . . . . . | 25 |
| 5. Request/Response Semantics . . . . .                           | 27 |
| 5.1. Requests . . . . .   | 27 |
| 5.2. Responses . . . . .  | 27 |

|          |  |    |
|----------|--|----|
| 5.2.1.   | Piggy-backed . . . . .                               | 28 |
| 5.2.2.   | Separate . . . . .                                   | 29 |
| 5.2.3.   | Non-Confirmable . . . . .                            | 30 |
| 5.3.     | Request/Response Matching . . . . .                  | 30 |
| 5.4.     | Options . . . . .                                    | 31 |
| 5.4.1.   | Critical/Elective . . . . .                          | 32 |
| 5.4.2.   | Length . . . . .                                     | 32 |
| 5.4.3.   | Default Values . . . . .                             | 32 |
| 5.4.4.   | Repeatable Options . . . . .                         | 33 |
| 5.4.5.   | Option Numbers . . . . .                             | 33 |
| 5.5.     | Payload . . . . .                                    | 33 |
| 5.5.1.   | Representation . . . . .                             | 33 |
| 5.5.2.   | Diagnostic Message . . . . .                         | 33 |
| 5.6.     | Caching . . . . .                                    | 34 |
| 5.6.1.   | Freshness Model . . . . .                            | 35 |
| 5.6.2.   | Validation Model . . . . .                           | 35 |
| 5.7.     | Proxying . . . . .                                   | 35 |
| 5.8.     | Method Definitions . . . . .                         | 37 |
| 5.8.1.   | GET . . . . .  | 37 |
| 5.8.2.   | POST . . . . .                                       | 37 |
| 5.8.3.   | PUT . . . . .  | 37 |
| 5.8.4.   | DELETE . . . . .                                     | 38 |
| 5.9.     | Response Code Definitions . . . . .                  | 38 |
| 5.9.1.   | Success 2.xx . . . . .                               | 38 |
| 5.9.2.   | Client Error 4.xx . . . . .                          | 39 |
| 5.9.3.   | Server Error 5.xx . . . . .                          | 41 |
| 5.10.    | Option Definitions . . . . .                         | 41 |
| 5.10.1.  | Token . . . . .                                      | 42 |
| 5.10.2.  | Uri-Host, Uri-Port, Uri-Path and Uri-Query . . . . . | 42 |
| 5.10.3.  | Proxy-Uri . . . . .                                  | 43 |
| 5.10.4.  | Content-Type . . . . .                               | 44 |
| 5.10.5.  | Accept . . . . .                                     | 44 |
| 5.10.6.  | Max-Age . . . . .                                    | 45 |
| 5.10.7.  | ETag . . . . .                                       | 45 |
| 5.10.8.  | Location-Path and Location-Query . . . . .           | 45 |
| 5.10.9.  | If-Match . . . . .                                   | 46 |
| 5.10.10. | If-None-Match . . . . .                              | 47 |
| 6.       | CoAP URIs . . . . .                                  | 47 |
| 6.1.     | coap URI Scheme . . . . .                            | 47 |
| 6.2.     | coaps URI Scheme . . . . .                           | 48 |
| 6.3.     | Normalization and Comparison Rules . . . . .         | 48 |
| 6.4.     | Decomposing URIs into Options . . . . .              | 49 |
| 6.5.     | Composing URIs from Options . . . . .                | 50 |
| 7.       | Discovery . . . . .                                  | 51 |
| 7.1.     | Service Discovery . . . . .                          | 51 |
| 7.2.     | Resource Discovery . . . . .                         | 52 |
| 7.2.1.   | 'ct' Attribute . . . . .                             | 52 |
| 8.       | Multicast CoAP . . . . .                             | 53 |

|  |    |
|--|----|
| 8.1. Messaging Layer . . . . .                                   | 53 |
| 8.2. Request/Response Layer . . . . .                            | 53 |
| 8.2.1. Caching . . . . .   | 54 |
| 8.2.2. Proxying . . . . .  | 54 |
| 9. Securing CoAP . . . . .                                       | 54 |
| 9.1. DTLS-secured CoAP . . . . .                                 | 56 |
| 9.1.1. Messaging Layer . . . . .                                 | 57 |
| 9.1.2. Request/Response Layer . . . . .                          | 57 |
| 9.1.3. Endpoint Identity . . . . .                               | 58 |
| 9.2. Using CoAP with IPsec . . . . .                             | 60 |
| 10. Cross-Protocol Proxying between CoAP and HTTP . . . . .      | 60 |
| 10.1. CoAP-HTTP Mapping . . . . .                                | 61 |
| 10.1.1. GET . . . . .  | 62 |
| 10.1.2. PUT . . . . .  | 62 |
| 10.1.3. DELETE . . . . .   | 63 |
| 10.1.4. POST . . . . .   | 63 |
| 10.2. HTTP-CoAP Mapping . . . . .                                | 63 |
| 10.2.1. OPTIONS and TRACE . . . . .                              | 63 |
| 10.2.2. GET . . . . .  | 64 |
| 10.2.3. HEAD . . . . .   | 64 |
| 10.2.4. POST . . . . .   | 64 |
| 10.2.5. PUT . . . . .  | 65 |
| 10.2.6. DELETE . . . . .   | 65 |
| 10.2.7. CONNECT . . . . .  | 65 |
| 11. Security Considerations . . . . .                            | 65 |
| 11.1. Protocol Parsing, Processing URIs . . . . .                | 65 |
| 11.2. Proxying and Caching . . . . .                             | 66 |
| 11.3. Risk of amplification . . . . .                            | 66 |
| 11.4. IP Address Spoofing Attacks . . . . .                      | 67 |
| 11.5. Cross-Protocol Attacks . . . . .                           | 68 |
| 12. IANA Considerations . . . . .                                | 70 |
| 12.1. CoAP Code Registry . . . . .                               | 70 |
| 12.1.1. Method Codes . . . . .                                   | 70 |
| 12.1.2. Response Codes . . . . .                                 | 71 |
| 12.2. Option Number Registry . . . . .                           | 73 |
| 12.3. Media Type Registry . . . . .                              | 75 |
| 12.4. URI Scheme Registration . . . . .                          | 76 |
| 12.5. Secure URI Scheme Registration . . . . .                   | 77 |
| 12.6. Service Name and Port Number Registration . . . . .        | 78 |
| 12.7. Secure Service Name and Port Number Registration . . . . . | 79 |
| 12.8. Multicast Address Registration . . . . .                   | 79 |
| 13. Acknowledgements . . . . .                                   | 80 |
| 14. References . . . . .   | 80 |
| 14.1. Normative References . . . . .                             | 80 |
| 14.2. Informative References . . . . .                           | 83 |
| Appendix A. Examples . . . . .                                   | 84 |
| Appendix B. URI Examples . . . . .                               | 90 |
| Appendix C. Changelog . . . . .                                  | 91 |

Authors' Addresses . . . . . 99

## 1. Introduction

The use of web services on the Internet has become ubiquitous in most applications, and depends on the fundamental Representational State Transfer [REST] architecture of the web.

The Constrained RESTful Environments (CoRE) work aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN, [RFC4944]). Constrained networks like 6LoWPAN support the expensive fragmentation of IPv6 packets into small link-layer frames. One design goal of CoAP has been to keep message overhead small, thus limiting the use of fragmentation.

One of the main goals of CoAP is to design a generic web protocol for the special requirements of this constrained environment, especially considering energy, building automation and other machine-to-machine (M2M) applications. The goal of CoAP is not to blindly compress HTTP [RFC2616], but rather to realize a subset of REST common with HTTP but optimized for M2M applications. Although CoAP could be used for compressing simple HTTP interfaces, it more importantly also offers features for M2M such as built-in discovery, multicast support and asynchronous message exchanges.

This document specifies the Constrained Application Protocol (CoAP), which easily translates to HTTP for integration with the existing web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments and M2M applications.

### 1.1. Features

CoAP has the following main features:

- o Constrained web protocol fulfilling M2M requirements.
- o UDP binding with optional reliability supporting unicast and multicast requests.
- o Asynchronous message exchanges.
- o Low header overhead and parsing complexity.
- o URI and Content-type support.
- o Simple proxy and caching capabilities.

- o A stateless HTTP mapping, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP.
- o Security binding to Datagram Transport Layer Security (DTLS).

## 1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. These words may also appear in this document in lower case as plain English words, absent their normative meanings.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC2616]. In addition, this specification defines the following terminology:

### Endpoint

An entity participating in the CoAP protocol. Colloquially, an endpoint lives on a "Node", although "Host" would be more consistent with Internet standards usage, and is further identified by transport layer multiplexing information that can include a UDP port number and a security association (Section 4.1).

### Sender

The originating endpoint of a message. When the aspect of identification of the specific sender is in focus, also "source endpoint".

### Recipient

The destination endpoint of a message. When the aspect of identification of the specific recipient is in focus, also "destination endpoint".

### Client

The originating endpoint of a request; the destination endpoint of a response.

### Server

The destination endpoint of a request; the originating endpoint of a response.

### Origin Server

The server on which a given resource resides or is to be created.

### Intermediary

A CoAP endpoint that acts both as a server and as a client towards (possibly via further intermediaries) an origin server. There are two common forms of intermediary: proxy and reverse proxy. In some cases, a single endpoint might act as an origin server, proxy, or reverse proxy, switching behavior based on the nature of each request.

### Proxy

A "proxy" is an endpoint selected by a client, usually via local configuration rules, to perform requests on behalf of the client, doing any necessary translations. Some translations are minimal, such as for proxy requests for "coap" URIs, whereas other requests might require translation to and from entirely different application-layer protocols.

### Reverse Proxy

A "reverse proxy" is an endpoint that acts as a layer above some other server(s) and satisfies requests on behalf of these, doing any necessary translations. Unlike a proxy, a reverse proxy receives requests as if it was the origin server for the target resource; the requesting client will not be aware that it is communicating with a reverse proxy.

### Confirmable Message

Some messages require an acknowledgement. These messages are called "Confirmable". When no packets are lost, each confirmable message elicits exactly one return message of type Acknowledgement or type Reset.

### Non-Confirmable Message

Some other messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.

### Acknowledgement Message

An Acknowledgement message acknowledges that a specific Confirmable Message arrived. It does not indicate success or failure of any encapsulated request.

### Reset Message

A Reset message indicates that a specific message (confirmable or non-confirmable) was received, but some context is missing to properly process it. This condition is usually caused when the receiving node has rebooted and has forgotten some state that would be required to interpret the message.

#### Piggy-backed Response

A Piggy-backed Response is included right in a CoAP Acknowledgement (ACK) message that is sent to acknowledge receipt of the Request for this Response (Section 5.2.1).

#### Separate Response

When a Confirmable message carrying a Request is acknowledged with an empty message (e.g., because the server doesn't have the answer right away), a Separate Response is sent in a separate message exchange (Section 5.2.2).

#### Critical Option

An option that would need to be understood by the endpoint receiving the message in order to properly process the message (Section 5.4.1). Note that the implementation of critical options is, as the name "Option" implies, generally optional: unsupported critical options lead to rejection of the message.

#### Elective Option

An option that is intended to be ignored by an endpoint that does not understand it. Processing the message even without understanding the option is acceptable (Section 5.4.1).

#### Resource Discovery

The process where a CoAP client queries a server for its list of hosted resources (i.e., links, Section 7).

In this specification, the term "byte" is used in its now customary sense as a synonym for "octet".

In this specification, the operator "^" stands for exponentiation.

## 2. Constrained Application Protocol

The interaction model of CoAP is similar to the client/server model of HTTP. However, machine-to-machine interactions typically result in a CoAP implementation acting in both client and server roles. A CoAP request is equivalent to that of HTTP, and is sent by a client to request an action (using a method code) on a resource (identified by a URI) on a server. The server then sends a response with a response code; this response may include a resource representation.

Unlike HTTP, CoAP deals with these interchanges asynchronously over a datagram-oriented transport such as UDP. This is done logically using a layer of messages that supports optional reliability (with exponential back-off). CoAP defines four types of messages: Confirmable, Non-Confirmable, Acknowledgement, Reset; method codes



and response codes included in some of these messages make them carry requests or responses. The basic exchanges of the four types of messages are somewhat orthogonal to the request/response interactions; requests can be carried in Confirmable and Non-Confirmable messages, and responses can be carried in these as well as piggy-backed in acknowledgements.

One could think of CoAP logically as using a two-layer approach, a CoAP messaging layer used to deal with UDP and the asynchronous nature of the interactions, and the request/response interactions using Method and Response codes (see Figure 1). CoAP is however a single protocol, with messaging and request/response just features of the CoAP header.

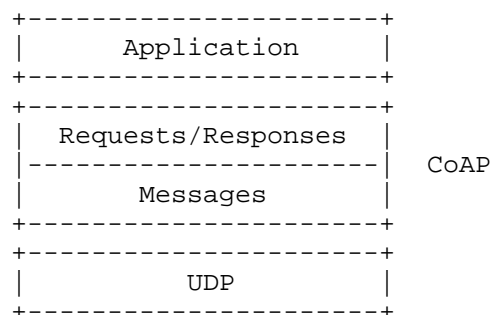


Figure 1: Abstract layering of CoAP

### 2.1. Messaging Model

The CoAP messaging model is based on the exchange of messages over UDP between endpoints.

CoAP uses a short fixed-length binary header (4 bytes) that may be followed by compact binary options and a payload. This message format is shared by requests and responses. The CoAP message format is specified in Section 3. Each message contains a Message ID used to detect duplicates and for optional reliability.

Reliability is provided by marking a message as Confirmable (CON). A Confirmable message is retransmitted using a default timeout and exponential back-off between retransmissions, until the recipient sends an Acknowledgement message (ACK) with the same Message ID (for example, 0x7d34) from the corresponding endpoint; see Figure 2. When a recipient is not able to process a Confirmable message, it replies with a Reset message (RST) instead of an Acknowledgement (ACK).

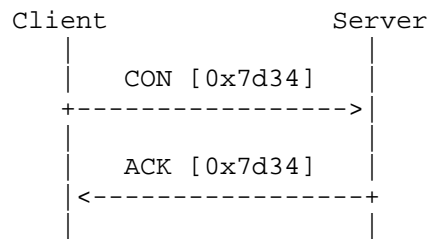


Figure 2: Reliable message transmission

A message that does not require reliable transmission, for example each single measurement out of a stream of sensor data, can be sent as a Non-confirmable message (NON). These are not acknowledged, but still have a Message ID for duplicate detection; see Figure 3. When a recipient is not able to process a Non-confirmable message, it may reply with a Reset message (RST).

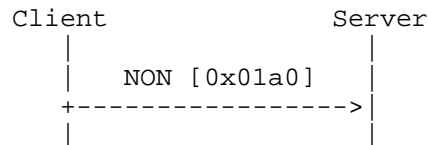


Figure 3: Unreliable message transmission

See Section 4 for details of CoAP messages.

As CoAP is based on UDP, it also supports the use of multicast IP destination addresses, enabling multicast CoAP requests. Section 8 discusses the proper use of CoAP messages with multicast addresses and precautions for avoiding response congestion.

Several security modes are defined for CoAP in Section 9 ranging from no security to certificate-based security. The use of IPsec along with a binding to DTLS are specified for securing the protocol.

## 2.2. Request/Response Model

CoAP request and response semantics are carried in CoAP messages, which include either a method code or response code, respectively. Optional (or default) request and response information, such as the URI and payload content-type are carried as CoAP options. A Token Option is used to match responses to requests independently from the underlying messages (Section 5.3).

A request is carried in a Confirmable (CON) or Non-confirmable (NON) message, and if immediately available, the response to a request

carried in a Confirmable message is carried in the resulting Acknowledgement (ACK) message. This is called a piggy-backed response, detailed in Section 5.2.1. Two examples for a basic GET request with piggy-backed response are shown in Figure 4, one successful, one resulting in a 4.04 (Not Found) response.

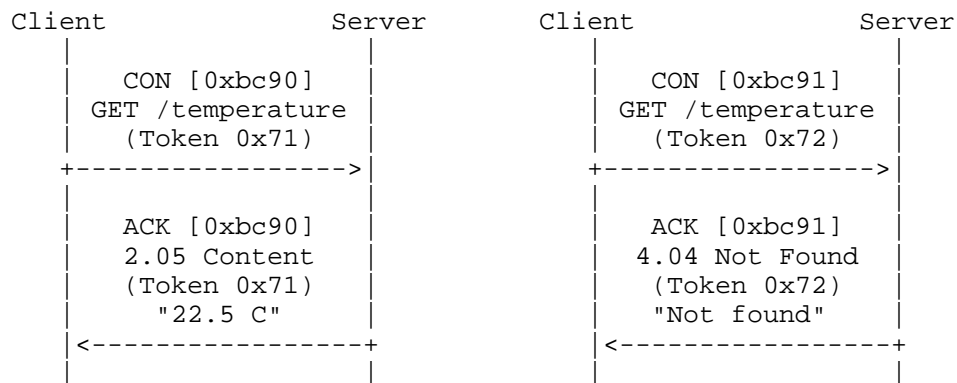


Figure 4: Two GET requests with piggy-backed responses

If the server is not able to respond immediately to a request carried in a Confirmable message, it simply responds with an empty Acknowledgement message so that the client can stop retransmitting the request. When the response is ready, the server sends it in a new Confirmable message (which then in turn needs to be acknowledged by the client). This is called a separate response, as illustrated in Figure 5 and described in more detail in Section 5.2.2.

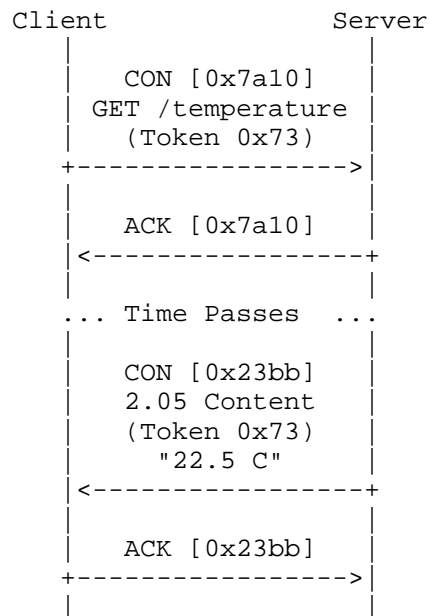


Figure 5: A GET request with a separate response

Likewise, if a request is sent in a Non-Confirmable message, then the response is usually sent using a new Non-Confirmable message, although the server may send a Confirmable message. This type of exchange is illustrated in Figure 6.

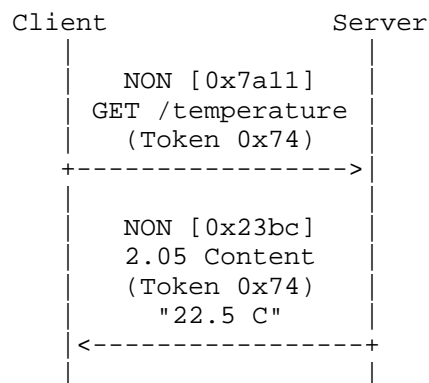


Figure 6: A NON request and response

CoAP makes use of GET, PUT, POST and DELETE methods in a similar manner to HTTP, with the semantics specified in Section 5.8. (Note that the detailed semantics of CoAP methods are "almost, but not

entirely unlike" those of HTTP methods: Intuition taken from HTTP experience generally does apply well, but there are enough differences that make it worthwhile to actually read the present specification.)

URI support in a server is simplified as the client already parses the URI and splits it into host, port, path and query components, making use of default values for efficiency. Response codes correspond to a small subset of HTTP response codes with a few CoAP specific codes added, as defined in Section 5.9.

### 2.3. Intermediaries and Caching

The protocol supports the caching of responses in order to efficiently fulfill requests. Simple caching is enabled using freshness and validity information carried with CoAP responses. A cache could be located in an endpoint or an intermediary. Caching functionality is specified in Section 5.6.

Proxying is useful in constrained networks for several reasons, including network traffic limiting, to improve performance, to access resources of sleeping devices or for security reasons. The proxying of requests on behalf of another CoAP endpoint is supported in the protocol. When using a proxy, the URI of the resource to request is included in the request, while the destination IP address is set to the address of the proxy. See Section 5.7 for more information on proxy functionality.

As CoAP was designed according to the REST architecture and thus exhibits functionality similar to that of the HTTP protocol, it is quite straightforward to map from CoAP to HTTP and from HTTP to CoAP. Such a mapping may be used to realize an HTTP REST interface using CoAP, or for converting between HTTP and CoAP. This conversion can be carried out by a proxy, which converts the method or response code, content-type, and options to the corresponding HTTP feature. Section 10 provides more detail about HTTP mapping.

### 2.4. Resource Discovery

Resource discovery is important for machine-to-machine interactions, and is supported using the CoRE Link Format [I-D.ietf-core-link-format] as discussed in Section 7.

## 3. Message Format

CoAP is based on the exchange of short messages which, by default, are transported over UDP (i.e. each CoAP message occupies the data

section of one UDP datagram). CoAP may be used with Datagram Transport Layer Security (DTLS) (see Section 9.1). It could also be used over other transports such as TCP or SCTP, the specification of which is out of this document's scope.

CoAP messages are encoded in a simple binary format. A message consists of a fixed-sized CoAP Header followed by options in Type-Length-Value (TLV) format and a payload. The number of options is determined by the header. The payload is made up of the bytes after the options, if any; its length is calculated from the datagram length.

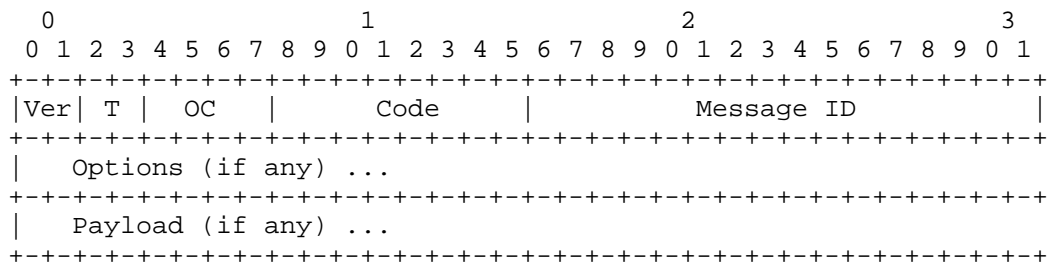


Figure 7: Message Format

### 3.1. Header Format

The fields in the header are defined as follows:

Version (Ver): 2-bit unsigned integer. Indicates the CoAP version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.

Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-Confirmable (1), Acknowledgement (2) or Reset (3). See Section 4 for the semantics of these message types.

Option Count (OC): 4-bit unsigned integer. Indicates the number of options after the header (0-14). If set to 0, there are no options and the payload (if any) immediately follows the header. If set to 15, then an end-of-options marker is used to indicate the end of options and the start of the payload. The format of options is defined below.

Code: 8-bit unsigned integer. Indicates if the message carries a request (1-31) or a response (64-191), or is empty (0). (All other code values are reserved.) In case of a request, the Code field indicates the Request Method; in case of a response a

Response Code. Possible values are maintained in the CoAP Code Registry (Section 12.1). See Section 5 for the semantics of requests and responses.

Message ID: 16-bit unsigned integer in network byte order. Used for the detection of message duplication, and to match messages of type Acknowledgement/Reset and messages of type Confirmable/Non-confirmable. See Section 4 for Message ID generation rules and how messages are matched.

### 3.2. Option Format

Options MUST appear in order of their Option Number (see Section 5.4.5). A delta encoding is used between options: The Option Number for each Option is calculated as the sum of its Option Delta field and the Option Number of the preceding Option in the message, if any. For the first Option in the message, the Option Delta becomes the Option Number (i.e., an implementation can simply initialize the number variable as zero). Multiple options with the same Option Number can be included by using an Option Delta of zero. Following the Option Delta, each option has a Length field which specifies the length of the Option Value, in bytes. The Length field can be extended by one byte for options with values longer than 14 bytes. The Option Value immediately follows the Length field.

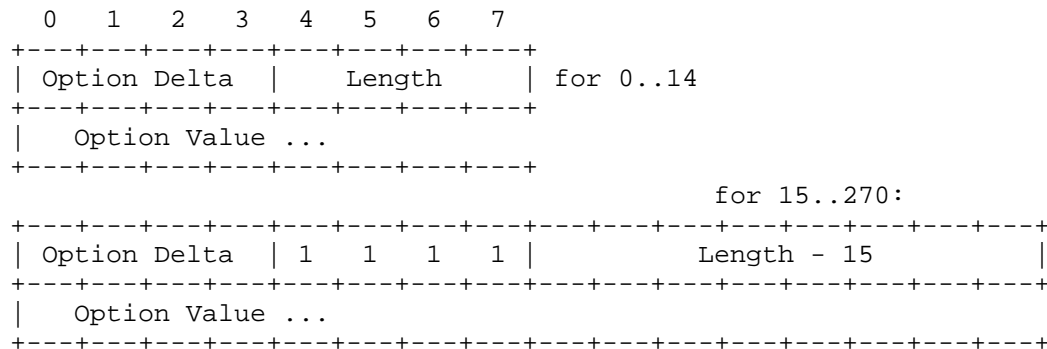


Figure 8: Option Format

The fields in an option are defined as follows:

Option Delta: 4-bit unsigned integer. Indicates the difference between the Option Number of this option and the previous option (or zero for the first option). In other words, the Option Number is calculated by simply summing the Option Delta fields of this and previous options before it. If a delta larger than 14 is needed, the Option Numbers that are non-zero multiples of 14

(i.e., 14, 28, 42, ...) can be used with the Length field set to 0 as "fenceposts". The Option Delta 15 is reserved for the the end-of-options marker (see below).

**Length:** Indicates the length of the Option Value, in bytes.

Normally Length is a 4-bit unsigned integer allowing value lengths of 0-14 bytes. When the Length field is set to 15, another byte is added as an 8-bit unsigned integer whose value is added to the 15, allowing option value lengths of 15-270 bytes.

**Value:** The length and format of the Option Value depends on the respective option, which MAY define variable length values. See Section 3.3 for the formats the options defined in this document make use of; other options MAY make use of other option value formats.

If the Option Count field in the header is 15 and the Option Delta is 15, the option is interpreted as the end-of-options marker instead of the option with the resulting Option Number. A sender MUST NOT include a value with the marker (i.e., the option length is 0) and a recipient MUST ignore any value of the marker. When this marker is encountered, it is immediately followed by the payload (if any). (Note that, by this special meaning, the Option Delta of 15 is made special, not any specific Option Number.) The sender MUST NOT include the Option Delta of 15 in a message with an Option Count other than 15.

Option Numbers are maintained in the CoAP Option Number Registry (Section 12.2). See Section 5.10 for the semantics of the options defined in this document.

### 3.3. Option Value Formats

The options defined in this document make use of the following option value formats.

#### 3.3.1. uint

A non-negative integer which is represented in network byte order using the given number of bytes. An option definition may specify a range of permissible numbers of bytes; if it has a choice, a sender SHOULD represent the integer with as few bytes as possible, i.e., without leading zeros. A recipient MUST be prepared to process values with leading zeros.



Implementation Note: The exceptional behavior permitted above is for highly constrained templated implementations (e.g. hardware implementations) that use fixed size options in the templates.

Length = 0            (implies value of 0)

```

      0
      0 1 2 3 4 5 6 7
+-----+
Length = 1 |      0-255      |
+-----+
```

```

      0                               1
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-----+
Length = 2 |      0-65535      |
+-----+
```

Length = 3 is 24 bits, Length = 4 is 32 bits etc.

### 3.3.2. string

A Unicode string which is encoded using UTF-8 [RFC3629] in Net-Unicode form [RFC5198]. Note that here and in all other places where UTF-8 encoding is used in the CoAP protocol, the intention is that the encoded strings can be directly used and compared as opaque byte strings by CoAP protocol implementations. There is no expectation and no need to perform normalization within a CoAP implementation unless Unicode strings that are not known to be normalized are imported from sources outside the CoAP protocol. Note also that ASCII strings (that do not make use of special control characters) are always valid UTF-8 Net-Unicode strings.

### 3.3.3. opaque

An opaque sequence of bytes.

### 3.3.4. empty

A zero-length sequence of bytes.

## 4. Message Transmission

CoAP messages are exchanged asynchronously between CoAP endpoints. They are used to transport CoAP requests and responses, the semantics of which are defined in Section 5.

As CoAP is bound to non-reliable transports such as UDP, CoAP messages may arrive out of order, appear duplicated, or go missing without notice. For this reason, CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport like TCP. It has the following features:

- o Simple stop-and-wait retransmission reliability with exponential back-off for "confirmable" messages.
- o Duplicate detection for both "confirmable" and "non-confirmable" messages.

#### 4.1. Messages and Endpoints

A CoAP endpoint is the source or destination of a CoAP message. It is identified depending on the security mode used (see Section 9): With no security, the endpoint is solely identified by an IP address and a UDP port number. With other security modes, the endpoint is identified as defined by the security mode.

There are different types of messages. The type of a message is specified by the T field of the CoAP header.

Separate from the message type, a message may carry a request, a response, or be empty. This is signaled by the Code field in the CoAP header and is relevant to the request/response model. Possible values for the Code field are maintained by the CoAP Code Registry (Section 12.1).

An empty message has the Code field set to 0. The OC field SHOULD be set to 0 and no bytes SHOULD be present after the Message ID field. The OC field and any bytes trailing the header MUST be ignored by any recipient.

#### 4.2. Messages Transmitted Reliably

The reliable transmission of a message is initiated by marking the message as "confirmable" in the CoAP header. A confirmable message always carries either a request or response and MUST NOT be empty. A recipient MUST acknowledge such a message with an acknowledgement message or, if it lacks context to process the message properly, MUST reject it with a reset message. The acknowledgement message MUST echo the Message ID of the confirmable message, and MUST carry a response or be empty (see Section 5.2.1 and Section 5.2.2). The reset message MUST echo the Message ID of the confirmable message, and MUST be empty.

The sender retransmits the confirmable message at exponentially

increasing intervals, until it receives an acknowledgement (or reset message), or runs out of attempts.

Retransmission is controlled by two things that a CoAP endpoint **MUST** keep track of for each confirmable message it sends while waiting for an acknowledgement (or reset): a timeout and a retransmission counter. For a new confirmable message, the initial timeout is set to a random number between `ACK_TIMEOUT` and `(ACK_TIMEOUT * ACK_RANDOM_FACTOR)` (see Section 4.8), and the retransmission counter is set to 0. When the timeout is triggered and the retransmission counter is less than `MAX_RETRANSMIT`, the message is retransmitted, the retransmission counter is incremented, and the timeout is doubled. If the retransmission counter reaches `MAX_RETRANSMIT` on a timeout, or if the endpoint receives a reset message, then the attempt to transmit the message is canceled and the application process informed of failure. On the other hand, if the endpoint receives an acknowledgement message in time, transmission is considered successful.

A CoAP endpoint that sent a confirmable message **MAY** give up in attempting to obtain an ACK even before the `MAX_RETRANSMIT` counter value is reached: E.g., the application has canceled the request as it no longer needs a response, or there is some other indication that the CON message did arrive. In particular, a CoAP request message may have elicited a separate response, in which case it is clear to the requester that only the ACK was lost and a retransmission of the request would serve no purpose. However, a responder **MUST NOT** in turn rely on this cross-layer behavior from a requester, i.e. it **SHOULD** retain the state to create the ACK for the request, if needed, even if a confirmable response was already acknowledged by the requester.

#### 4.3. Messages Transmitted Without Reliability

Some messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor where eventual success is sufficient.

As a more lightweight alternative, a message can be transmitted less reliably by marking the message as "non-confirmable". A non-confirmable message always carries either a request or response and **MUST NOT** be empty. A non-confirmable message **MUST NOT** be acknowledged by the recipient. If a recipient lacks context to process the message properly, it **MAY** reject the message with a reset message or otherwise **MUST** silently ignore it.

At the CoAP level, there is no way for the sender to detect if a non-

confirmable message was received or not. A sender MAY choose to transmit a non-confirmable message multiple times, or the network may duplicate the message in transit. To enable the receiver to act only once on the message, non-confirmable messages specify a Message ID as well. (This Message ID is drawn from the same number space as the Message IDs for confirmable messages.)

#### 4.4. Message Correlation

An acknowledgement or reset message is related to a confirmable message or non-confirmable message by means of a Message ID along with additional address information of the corresponding endpoint. The Message ID is a 16-bit unsigned integer that is generated by the sender of a confirmable or non-confirmable message and included in the CoAP header. The Message ID MUST be echoed in the acknowledgement or reset message by the recipient.

The same Message ID MUST NOT be re-used (per Message ID variable) within the EXCHANGE\_LIFETIME (Section 4.8.2).

Implementation Note: Several implementation strategies can be employed for generating Message IDs. In the simplest case a CoAP endpoint generates Message IDs by keeping a single Message ID variable, which is changed each time a new confirmable or non-confirmable message is sent regardless of the destination address or port. Endpoints dealing with large numbers of transactions could keep multiple Message ID variables, for example per prefix or destination address. The initial variable value should be randomized.

For an acknowledgement or reset message to match a confirmable or non-confirmable message, the Message ID and source endpoint of the acknowledgement or reset message MUST match the Message ID and destination endpoint of the confirmable or non-confirmable message.

#### 4.5. Message Deduplication

A recipient MUST be prepared to receive the same confirmable message (as indicated by the Message ID and source endpoint) multiple times within the EXCHANGE\_LIFETIME (Section 4.8.2), for example, when its acknowledgement went missing or didn't reach the original sender before the first timeout. The recipient SHOULD acknowledge each duplicate copy of a confirmable message using the same acknowledgement or reset message, but SHOULD process any request or response in the message only once. This rule MAY be relaxed in case the confirmable message transports a request that is idempotent (see Section 5.1) or can be handled in an idempotent fashion. Examples for relaxed message deduplication:

- o A server MAY relax the requirement to answer all retransmissions of an idempotent request with the same response (Section 4.2), so that it does not have to maintain state for Message IDs. For example, an implementation might want to process duplicate transmissions of a GET, PUT or DELETE request as separate requests if the effort incurred by duplicate processing is less expensive than keeping track of previous responses would be.
- o A constrained server MAY even want to relax this requirement for certain non-idempotent requests if the application semantics make this trade-off favorable. For example, if the result of a POST request is just the creation of some short-lived state at the server, it may be less expensive to incur this effort multiple times for a request than keeping track of whether a previous transmission of the same request already was processed.

A recipient MUST be prepared to receive the same non-confirmable message (as indicated by the Message ID and source endpoint) multiple times within NON\_LIFETIME (Section 4.8.2). As a general rule that may be relaxed based on the specific semantics of a message, the recipient SHOULD silently ignore any duplicated non-confirmable message, and SHOULD process any request or response in the message only once.

#### 4.6. Message Size

While specific link layers make it beneficial to keep CoAP messages small enough to fit into their link layer packets (see Section 1), this is a matter of implementation quality. The CoAP specification itself provides only an upper bound to the message size. Messages larger than an IP fragment result in undesired packet fragmentation. A CoAP message, appropriately encapsulated, SHOULD fit within a single IP packet (i.e., avoid IP fragmentation) and (by fitting into one UDP payload) obviously MUST fit within a single IP datagram. If the Path MTU is not known for a destination, an IP MTU of 1280 bytes SHOULD be assumed; if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size.

Implementation Note: CoAP's choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. (However, with IPv4, it is harder to absolutely ensure that there is no IP fragmentation. If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; worse, the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload. Implementations extremely focused on this problem

set might also set the IPv4 DF bit and perform some form of path MTU discovery; this should generally be unnecessary in most realistic use cases for CoAP, however.) A more important kind of fragmentation in many constrained networks is that on the adaptation layer (e.g., 6LoWPAN L2 packets are limited to 127 bytes including various overheads); this may motivate implementations to be frugal in their packet sizes and to move to block-wise transfers [I-D.ietf-core-block] when approaching three-digit message sizes.

Message sizes are also of considerable importance to implementations on constrained nodes. Many implementations will need to allocate a buffer for incoming messages. If an implementation is too constrained to allow for allocating the above-mentioned upper bound, it could apply the following implementation strategy: Implementations receiving a datagram into a buffer that is too small are usually able to determine if the trailing portion of a datagram was discarded and to retrieve the initial portion. So, if not all of the payload, at least the CoAP header and options are likely to fit within the buffer. A server can thus fully interpret a request and return a 4.13 (Request Entity Too Large) response code if the payload was truncated. A client sending an idempotent request and receiving a response larger than would fit in the buffer can repeat the request with a suitable value for the Block Option [I-D.ietf-core-block].

#### 4.7. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2.

In order not to cause congestion, Clients (including proxies) SHOULD strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies). An outstanding interaction is either a CON for which an ACK has not yet been received but is still expected (message layer) or a request for which a response has not yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). A good value for this limit is the number 1. (Note that [RFC2616], in trying to achieve a similar objective, did specify a specific number of simultaneous connections as a ceiling. While revising [RFC2616], this was found to be impractical for many applications [I-D.ietf-httpbis-pl-messaging]. For the same considerations, this specification does not mandate a particular maximum number of outstanding interactions, but instead encourages clients to be conservative when initiating interactions.)

Further congestion control optimizations and considerations are

expected in the future, which may for example provide automatic initialization of the CoAP transmission parameters defined in Section 4.8.

#### 4.8. Transmission Parameters

Message transmission is controlled by the following parameters:

| name              | default value |
|-------------------|---------------|
| ACK_TIMEOUT       | 2 seconds     |
| ACK_RANDOM_FACTOR | 1.5           |
| MAX_RETRANSMIT    | 4             |

##### 4.8.1. Changing The Parameters

The values for ACK\_TIMEOUT, ACK\_RANDOM\_FACTOR, and MAX\_RETRANSMIT may be configured to values specific to the application environment, however the configuration method is out of scope of this document. It is recommended that an application environment use consistent values for these parameters.

The transmission parameters have been chosen to achieve a behavior in the presence of congestion that is safe in the Internet. If a configuration desires to use different values, the onus is on the configuration to ensure these congestion control properties are not violated. In particular, a decrease of ACK\_TIMEOUT below 1 second would violate the guidelines of [RFC5405].

([I-D.allman-tcpm-rto-consider] provides some additional background.) CoAP was designed to enable implementations that do not maintain round-trip-time (RTT) measurements. However, where it is desired to decrease the ACK\_TIMEOUT significantly, this can only be done safely when maintaining such measurements. Configurations MUST NOT decrease ACK\_TIMEOUT without using mechanisms that ensure congestion control safety, either defined in the configuration or in future standards documents.

ACK\_RANDOM\_FACTOR MUST NOT be decreased below 1.0, and it SHOULD have a value that is sufficiently different from 1.0 to provide some protection from synchronization effects.

MAX\_RETRANSMIT can be freely adjusted, but a too small value will reduce the probability that a confirmable message is actually received, while a larger value will require further adjustments in the time values (see discussion below).

If the choice of transmission parameters leads to an increase of derived time values (see below), the configuration mechanism MUST ensure the adjusted value is available to the corresponding end-points, too.

#### 4.8.2. Time Values derived from Transmission Parameters

The combination of `ACK_TIMEOUT`, `ACK_RANDOM_FACTOR` and `MAX_RETRANSMIT` influences the timing of retransmissions, which in turn influences how long certain information items need to be kept by an implementation. To be able to unambiguously reference these derived time values, we give them names as follows:

- o `MAX_TRANSMIT_SPAN` is the maximum time from the first transmission of a confirmable message to its last retransmission. For the default transmission parameters, the value is  $(2+4+8+16)*1.5 = 45$  seconds, or more generally:

$$\text{ACK\_TIMEOUT} * (2 ** \text{MAX\_RETRANSMIT} - 1) * \text{ACK\_RANDOM\_FACTOR}$$

- o `MAX_TRANSMIT_WAIT` is the maximum time from the first transmission of a confirmable message to the time when the sender gives up on receiving an acknowledgement or reset. For the default transmission parameters, the value is  $(2+4+8+16+32)*1.5 = 93$  seconds, or more generally:

$$\text{ACK\_TIMEOUT} * (2 ** (\text{MAX\_RETRANSMIT} + 1) - 1) * \text{ACK\_RANDOM\_FACTOR}$$

In addition, some assumptions need to be made on the characteristics of the network and the nodes.

- o `MAX_LATENCY` is the maximum time a datagram is expected to take from the start of its transmission to the completion of its reception. This constant is related to the MSL (Maximum Segment Lifetime) of [RFC0793], which is "arbitrarily defined to be 2 minutes" ([RFC0793] glossary, page 81). Note that this is not necessarily smaller than `MAX_TRANSMIT_WAIT`, as `MAX_LATENCY` is not intended to describe a situation when the protocol works well, but the worst case situation against which the protocol has to guard. We, also arbitrarily, define `MAX_LATENCY` to be 100 seconds. Apart from being reasonably realistic for the bulk of configurations as well as close to the historic choice for TCP, this value also allows message ID lifetime timers to be represented in 8 bits (when measured in seconds). In these calculations, there is no assumption that the direction of the transmission is irrelevant (i.e. that the network is symmetric), just that the same value can reasonably be used as a maximum value for both directions. If



that is not the case, the following calculations become only slightly more complex.

- o `PROCESSING_DELAY` is the time a node takes to turn around a confirmable message into an acknowledgement. We assume the node will attempt to send an ACK before having the sender time out, so as a conservative assumption we set it equal to `ACK_TIMEOUT`.
- o `MAX_RTT` is the maximum round-trip time, or:

$$2 * \text{MAX\_LATENCY} + \text{PROCESSING\_DELAY}$$

From these values, we can derive the following values relevant to the protocol operation:

- o `EXCHANGE_LIFETIME` is the time from starting to send a confirmable message to the time when an acknowledgement is no longer expected, i.e. message layer information about the message exchange can be purged. `EXCHANGE_LIFETIME` includes a `MAX_TRANSMIT_SPAN`, a `MAX_LATENCY` forward, `PROCESSING_DELAY`, and a `MAX_LATENCY` for the way back. Note that there is no need to consider `MAX_TRANSMIT_WAIT` if the configuration is chosen such that the last waiting period (`ACK_TIMEOUT * (2 ** MAX_RETRANSMIT)`) or the difference between `MAX_TRANSMIT_SPAN` and `MAX_TRANSMIT_WAIT` is less than `MAX_LATENCY` -- which is a likely choice, as `MAX_LATENCY` is a worst case value unlikely to be met in the real world. In this case, `EXCHANGE_LIFETIME` simplifies to:

$$(\text{ACK\_TIMEOUT} * (2 ** \text{MAX\_RETRANSMIT} - 1) * \text{ACK\_RANDOM\_FACTOR}) + (2 * \text{MAX\_LATENCY}) + \text{PROCESSING\_DELAY}$$

or 248 seconds with the default transmission parameters.

- o `NON_LIFETIME` is the time from sending a non-confirmable message to the time its message-ID can be safely reused. If multiple transmission of a NON message is not used, its value is `MAX_LATENCY`, or 100 seconds. However, a CoAP sender might send a NON message multiple times, in particular for multicast applications. While the period of re-use is not bounded by the specification, an expectation of reliable detection of duplication at the receiver is in the timescales of `MAX_TRANSMIT_SPAN`. Therefore, for this purpose, it is safer to use the value:

$$\text{MAX\_TRANSMIT\_SPAN} + \text{MAX\_LATENCY}$$

or 145 seconds with the default transmission parameters; however, an implementation that just wants to use a single timeout value for retiring message-IDs can safely use the larger value for

EXCHANGE\_LIFETIME.

## 5. Request/Response Semantics

CoAP operates under a similar request/response model as HTTP: a CoAP endpoint in the role of a "client" sends one or more CoAP requests to a "server", which services the requests by sending CoAP responses. Unlike HTTP, requests and responses are not sent over a previously established connection, but exchanged asynchronously over CoAP messages.

### 5.1. Requests

A CoAP request consists of the method to be applied to the resource, the identifier of the resource, a payload and Internet media type (if any), and optional meta-data about the request.

CoAP supports the basic methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP. They have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP (see Section 9.1 of [RFC2616]). The GET method is safe, therefore it MUST NOT take any other action on a resource other than retrieval. The GET, PUT and DELETE methods MUST be performed in such a way that they are idempotent. POST is not idempotent, because its effect is determined by the origin server and dependent on the target resource; it usually results in a new resource being created or the target resource being updated.

A request is initiated by setting the Code field in the CoAP header of a confirmable or a non-confirmable message to a Method Code and including request information.

The methods used in requests are described in detail in Section 5.8.

### 5.2. Responses

After receiving and interpreting a request, a server responds with a CoAP response, which is matched to the request by means of a client-generated token.

A response is identified by the Code field in the CoAP header being set to a Response Code. Similar to the HTTP Status Code, the CoAP Response Code indicates the result of the attempt to understand and satisfy the request. These codes are fully defined in Section 5.9. The Response Code numbers to be set in the Code field of the CoAP header are maintained in the CoAP Response Code Registry (Section 12.1.2).

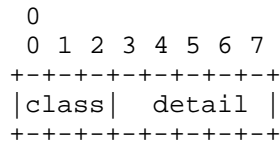


Figure 9: Structure of a Response Code

The upper three bits of the 8-bit Response Code number define the class of response. The lower five bits do not have any categorization role; they give additional detail to the overall class (Figure 9). There are 3 classes:

- 2 - Success: The request was successfully received, understood, and accepted.
- 4 - Client Error: The request contains bad syntax or cannot be fulfilled.
- 5 - Server Error: The server failed to fulfill an apparently valid request.

The response codes are designed to be extensible: Response Codes in the Client Error and Server Error class that are unrecognized by an endpoint MUST be treated as being equivalent to the generic Response Code of that class (4.00 and 5.00, respectively). However, there is no generic Response Code indicating success, so a Response Code in the Success class that is unrecognized by an endpoint can only be used to determine that the request was successful without any further details.

As a human readable notation for specifications and protocol diagnostics, the numeric value of a response code is indicated by giving the upper three bits in decimal, followed by a dot and then the lower five bits in a two-digit decimal. E.g., "Not Found" is written as 4.04 -- indicating a value of hexadecimal 0x84 or decimal 132. In other words, the dot "." functions as a short-cut for "\*32+".

The possible response codes are described in detail in Section 5.9.

Responses can be sent in multiple ways, which are defined below.

#### 5.2.1. Piggy-backed

In the most basic case, the response is carried directly in the acknowledgement message that acknowledges the request (which requires that the request was carried in a confirmable message). This is

called a "Piggy-backed" Response.

The response is returned in the acknowledgement message independent of whether the response indicates success or failure. In effect, the response is piggy-backed on the acknowledgement message, so no separate message is required to both acknowledge that the request was received and return the response.

Implementation note: The protocol leaves the decision whether to piggy-back a response or not (i.e., send a separate response) to the server. The client **MUST** be prepared to receive either. On the quality of implementation level, there is a strong expectation that servers will implement code to piggy-back whenever possible -- saving resources in the network and both at the client and at the server.

#### 5.2.2. Separate

It may not be possible to return a piggy-backed response in all cases. For example, a server might need longer to obtain the representation of the resource requested than it can wait sending back the acknowledgement message, without risking the client to repeatedly retransmit the request message. Responses to requests carried in a Non-Confirmable message are always sent separately (as there is no acknowledgement message).

The server maybe initiates the attempt to obtain the resource representation and times out an acknowledgement timer, or it immediately sends an acknowledgement knowing in advance that there will be no piggy-backed response. The acknowledgement effectively is a promise that the request will be acted upon.

When the server finally has obtained the resource representation, it sends the response. To ensure that this message is not lost, it is again sent as a confirmable message and answered by the client with an acknowledgement, echoing the new Message ID chosen by the server.

(Implementation notes: Note that, as the underlying datagram transport may not be sequence-preserving, the confirmable message carrying the response may actually arrive before or after the acknowledgement message for the request. Note also that, while the CoAP protocol itself does not make any specific demands here, there is an expectation that the response will come within a time frame that is reasonable from an application point of view; as there is no underlying transport protocol that could be instructed to run a keep-alive mechanism, the requester **MAY** want to set up a timeout that is unrelated to CoAP's retransmission timers in case the server is destroyed or otherwise unable to send the response.)

For a separate exchange, both the acknowledgement to the confirmable request and the acknowledgement to the confirmable response MUST be an empty message, i.e. one that carries neither a request nor a response.

#### 5.2.3. Non-Confirmable

If the request message is non-confirmable, then the response SHOULD be returned in a non-confirmable message as well. However, an endpoint MUST be prepared to receive a non-confirmable response (preceded or followed by an empty acknowledgement message) in reply to a confirmable request, or a confirmable response in reply to a non-confirmable request.

#### 5.3. Request/Response Matching

Regardless of how a response is sent, it is matched to the request by means of a token that is included by the client in the request as one of the options along with additional address information of the corresponding endpoint. The token MUST be echoed by the server in any resulting response without modification.

The exact rules for matching a response to a request are as follows:

1. The source endpoint of the response MUST be the same as the destination endpoint of the original request.
2. In a piggy-backed response, both the Message ID of the confirmable request and the acknowledgement, and the token of the response and original request MUST match. In a separate response, just the token of the response and original request MUST match.

The client SHOULD generate tokens in a way that tokens currently in use for a given source/destination pair are unique. (Note that a client can use the same token for any request if it uses a different source port number each time.)

An endpoint that did not generate a token MUST treat it as opaque and make no assumptions about its format. (Note that there is a default value for the Token Option, so every message carries a token, even if it is not explicitly expressed in a CoAP option.)

In case a message carrying a response is unexpected (i.e. the client is not waiting for a response with the specified address and/or token), the response SHOULD be rejected with a reset message and MUST NOT be acknowledged.

#### 5.4. Options

Both requests and responses may include a list of one or more options. For example, the URI in a request is transported in several options, and meta-data that would be carried in an HTTP header in HTTP is supplied as options as well.

CoAP defines a single set of options that are used in both requests and responses:

- o Content-Type
- o ETag
- o Location-Path
- o Location-Query
- o Max-Age
- o Proxy-Uri
- o Token
- o Uri-Host
- o Uri-Path
- o Uri-Port
- o Uri-Query
- o Accept
- o If-Match
- o If-None-Match

The semantics of these options along with their properties are defined in detail in Section 5.10.

Not all options are defined for use with all methods and response codes. The possible options for methods and response codes are defined in Section 5.8 and Section 5.9 respectively. In case an option is not defined for a method or response code, it **MUST NOT** be included by a sender and **MUST** be treated like an unrecognized option by a recipient.

#### 5.4.1. Critical/Elective

Options fall into one of two classes: "critical" or "elective". The difference between these is how an option unrecognized by an endpoint is handled:

- o Upon reception, unrecognized options of class "elective" MUST be silently ignored.
- o Unrecognized options of class "critical" that occur in a confirmable request MUST cause the return of a 4.02 (Bad Option) response. This response SHOULD include a diagnostic message describing the unrecognized option(s) (see Section 5.5.2).
- o Unrecognized options of class "critical" that occur in a confirmable response SHOULD cause the response to be rejected with a reset message.
- o Unrecognized options of class "critical" that occur in a non-confirmable message MUST cause the message to be silently ignored. The response MAY be rejected with a reset message.

Note that, whether critical or elective, an option is never "mandatory" (it is always optional): These rules are defined in order to enable implementations to reject options they do not understand or implement.

#### 5.4.2. Length

Option values are defined to have a specific length, often in the form of an upper and lower bound. If the length of an option value in a request is outside the defined range, that option MUST be treated like an unrecognized option (see Section 5.4.1).

#### 5.4.3. Default Values

Options may be defined to have a default value. If the value of option is intended to be this default value, the option SHOULD NOT be included in the message. If the option is not present, the default value MUST be assumed.

Where a critical option has a default value, this is chosen in such a way that the absence of the option in a message can be processed properly both by implementations unaware of the critical option and by implementations that interpret this absence as the presence of the default value for the option.

#### 5.4.4. Repeatable Options

The definition of an option MAY specify the option to be repeatable. An option that is repeatable MAY be included one or more times in a message. An option that is not repeatable MUST NOT be included more than once in a message.

If a message includes an option with more occurrences than the option is defined for, the additional option occurrences MUST be treated like an unrecognized option (see Section 5.4.1).

#### 5.4.5. Option Numbers

Options are identified by an option number. Odd numbers indicate a critical option, while even numbers indicate an elective option. (Note that this is not just a convention, it is a feature of the protocol: Whether an option is elective or critical is entirely determined by whether its option number is even or odd.)

The numbers that are non-zero multiples of 14 are used in conjunction with "fenceposting", as described in Section 3.2. Options with these numbers MUST have a zero-length default value.

The option numbers for the options defined in this document are listed in the CoAP Option Number Registry (Section 12.2).

### 5.5. Payload

Both requests and responses may include payload, depending on the method or response code respectively. If a method or response code is not defined to have a payload, then a sender MUST NOT include one, and a recipient MUST ignore it.

#### 5.5.1. Representation

The payload of requests or of responses indicating success is typically a representation of a resource or the result of the requested action. Its format is specified by the Internet media type given by the Content-Type Option. In the absence of this option, no default value is assumed and the format must be inferred by the application (e.g., from the application context or by "sniffing" the payload).

#### 5.5.2. Diagnostic Message

The payload of responses indicating a client or server error is a brief human-readable diagnostic message, explaining the error situation. This diagnostic message MUST be encoded using UTF-8



[RFC3629], more specifically using Net-Unicode form [RFC5198]. The Content-Type Option MUST NOT be included by the sender and MUST be treated like an unrecognized option by the recipient.

The message is similar to the Reason-Phrase on an HTTP status line. It is not intended for end-users but for software engineers that during debugging need to interpret it in the context of the present, English-language specification; therefore no mechanism for language tagging is needed or provided.

## 5.6. Caching

CoAP endpoints MAY cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests.

The goal of caching in CoAP is to reuse a prior response message to satisfy a current request. In some cases, a stored response can be reused without the need for a network request, reducing latency and network round-trips; a "freshness" mechanism is used for this purpose (see Section 5.6.1). Even when a new request is required, it is often possible to reuse the payload of a prior response to satisfy the request, thereby reducing network bandwidth usage; a "validation" mechanism is used for this purpose (see Section 5.6.2).

Unlike HTTP, the cacheability of CoAP responses does not depend on the request method, but the Response Code. The cacheability of each Response Code is defined along the Response Code definitions in Section 5.9. Response Codes that indicate success and are unrecognized by an endpoint MUST NOT be cached.

For a presented request, a CoAP endpoint MUST NOT use a stored response, unless:

- o the presented request method and that used to obtain the stored response match,
- o all options match between those in the presented request and those of the request used to obtain the stored response (which includes the request URI), except that there is no need for a match of the Token, Max-Age, or ETag request option(s), and
- o the stored response is either fresh or successfully validated as defined below.

#### 5.6.1. Freshness Model

When a response is "fresh" in the cache, it can be used to satisfy subsequent requests without contacting the origin server, thereby improving efficiency.

The mechanism for determining freshness is for an origin server to provide an explicit expiration time in the future, using the Max-Age Option (see Section 5.10.6). The Max-Age Option indicates that the response is to be considered not fresh after its age is greater than the specified number of seconds.

The Max-Age Option defaults to a value of 60. Thus, if it is not present in a cacheable response, then the response is considered not fresh after its age is greater than 60 seconds. If an origin server wishes to prevent caching, it **MUST** explicitly include a Max-Age Option with a value of zero seconds.

#### 5.6.2. Validation Model

When an endpoint has one or more stored responses for a GET request, but cannot use any of them (e.g., because they are not fresh), it can use the ETag Option (Section 5.10.7) in the GET request to give the origin server an opportunity to both select a stored response to be used, and to update its freshness. This process is known as "validating" or "revalidating" the stored response.

When sending such a request, the endpoint **SHOULD** add an ETag Option specifying the entity-tag of each stored response that is applicable.

A 2.03 (Valid) response indicates the stored response identified by the entity-tag given in the response's ETag Option can be reused, after updating its freshness with the value of the Max-Age Option that is included with the response (see Section 5.9.1.3).

Any other response code indicates that none of the stored responses nominated in the request is suitable. Instead, the response **SHOULD** be used to satisfy the request and **MAY** replace the stored response.

#### 5.7. Proxying

CoAP distinguishes between requests to an origin server and a request made through a proxy. A proxy is a CoAP endpoint that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.

CoAP requests to a proxy are made as normal confirmable or non-confirmable requests to the proxy endpoint, but specify the request URI in a different way: The request URI in a proxy request is specified as a string in the Proxy-Uri Option (see Section 5.10.3), while the request URI in a request to an origin server is split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options (see Section 5.10.2).

When a proxy request is made to an endpoint and the endpoint is unwilling or unable to act as proxy for the request URI, it MUST return a 5.05 (Proxying Not Supported) response. If the authority (host and port) is recognized as identifying the proxy endpoint, then the request MUST be treated as a local request.

Unless a proxy is configured to forward the proxy request to another proxy, it MUST translate the request as follows: The origin server's IP address and port are determined by the authority component of the request URI, and the request URI is decoded and split into the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options.

All options present in a proxy request MUST be processed at the proxy. Critical options in a request that are not recognized by the proxy MUST lead to a 4.02 (Bad Option) response being returned by the proxy. Elective options not recognized by the proxy MUST NOT be forwarded to the origin server. Similarly, critical options in a response that are not recognized by the proxy server MUST lead to a 5.02 (Bad Gateway) response. Again, elective options that are not recognized MUST NOT be forwarded.

If the proxy does not employ a cache, then it simply forwards the translated request to the determined destination. Otherwise, if it does employ a cache but does not have a stored response that matches the translated request and is considered fresh, then it needs to refresh its cache according to Section 5.6.

If the request to the destination times out, then a 5.04 (Gateway Timeout) response MUST be returned. If the request to the destination returns an response that cannot be processed by the proxy, then a 5.02 (Bad Gateway) response MUST be returned. Otherwise, the proxy returns the response to the client.

If a response is generated out of a cache, it MUST be generated with a Max-Age Option that does not extend the max-age originally set by the server, considering the time the resource representation spent in the cache. E.g., the Max-Age Option could be adjusted by the proxy for each response using the formula:  $\text{proxy-max-age} = \text{original-max-age} - \text{cache-age}$ . For example if a request is made to a proxied resource that was refreshed 20 seconds ago and had an original Max-Age of 60

seconds, then that resource's proxied max-age is now 40 seconds.

## 5.8. Method Definitions

In this section each method is defined along with its behavior. A request with an unrecognized or unsupported Method Code MUST generate a 4.05 (Method Not Allowed) response.

### 5.8.1. GET

The GET method retrieves a representation for the information that currently corresponds to the resource identified by the request URI. If the request includes one or more Accept Options, they indicate the preferred content-type of a response. If the request includes an ETag Option, the GET method requests that ETag be validated and that the representation be transferred only if validation failed. Upon success a 2.05 (Content) or 2.03 (Valid) response SHOULD be sent.

The GET method is safe and idempotent.

### 5.8.2. POST

The POST method requests that the representation enclosed in the request be processed. The actual function performed by the POST method is determined by the origin server and dependent on the target resource. It usually results in a new resource being created or the target resource being updated.

If a resource has been created on the server, the response returned by the server SHOULD have a 2.01 (Created) response code and SHOULD include the URI of the new resource in a sequence of one or more Location-Path and/or Location-Query Options (Section 5.10.8). If the POST succeeds but does not result in a new resource being created on the server, the response SHOULD have a 2.04 (Changed) response code. If the POST succeeds and results in the target resource being deleted, the response SHOULD have a 2.02 (Deleted) response code.

POST is neither safe nor idempotent.

### 5.8.3. PUT

The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation. The representation format is specified by the media type given in the Content-Type Option.

If a resource exists at the request URI the enclosed representation SHOULD be considered a modified version of that resource, and a 2.04

(Changed) response SHOULD be returned. If no resource exists then the server MAY create a new resource with that URI, resulting in a 2.01 (Created) response. If the resource could not be created or modified, then an appropriate error response code SHOULD be sent.

Further restrictions to a PUT can be made by including the If-Match (see Section 5.10.9) or If-None-Match (see Section 5.10.10) options in the request.

PUT is not safe, but is idempotent.

#### 5.8.4. DELETE

The DELETE method requests that the resource identified by the request URI be deleted. A 2.02 (Deleted) response SHOULD be sent on success or in case the resource did not exist before the request.

DELETE is not safe, but is idempotent.

### 5.9. Response Code Definitions

Each response code is described below, including any options required in the response. Where appropriate, some of the codes will be specified in regards to related response codes in HTTP [RFC2616]; this does not mean that any such relationship modifies the HTTP mapping specified in Section 10.

#### 5.9.1. Success 2.xx

This class of status code indicates that the clients request was successfully received, understood, and accepted.

##### 5.9.1.1. 2.01 Created

Like HTTP 201 "Created", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

If the response includes one or more Location-Path and/or Location-Query Options, the values of these options specify the location at which the resource was created. Otherwise, the resource was created at the request URI. A cache MUST mark any stored response for the created resource as not fresh.

This response is not cacheable.

## 5.9.1.2.    2.02 Deleted

Like HTTP 204 "No Content", but only used in response to DELETE requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache SHOULD mark any stored response for the deleted resource as not fresh.

## 5.9.1.3.    2.03 Valid

Related to HTTP 304 "Not Modified", but only used to indicate that the response identified by the entity-tag identified by the included ETag Option is valid. Accordingly, the response MUST include an ETag Option.

When a cache receives a 2.03 (Valid) response, it MUST update the stored response with the value of the Max-Age Option included in the response (see Section 5.6.2).

## 5.9.1.4.    2.04 Changed

Like HTTP 204 "No Content", but only used in response to POST and PUT requests. The payload returned with the response, if any, is a representation of the action result.

This response is not cacheable. However, a cache MUST mark any stored response for the changed resource as not fresh.

## 5.9.1.5.    2.05 Content

Like HTTP 200 "OK", but only used in response to GET requests.

The payload returned with the response is a representation of the target resource.

This response is cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1) and (if present) the ETag Option for validation (see Section 5.6.2).

## 5.9.2.    Client Error 4.xx

This class of response code is intended for cases in which the client seems to have erred. These response codes are applicable to any request method.

The server SHOULD include a diagnostic message as detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

#### 5.9.2.1. 4.00 Bad Request

Like HTTP 400 "Bad Request".

#### 5.9.2.2. 4.01 Unauthorized

The client is not authorized to perform the requested action. The client SHOULD NOT repeat the request without previously improving its authentication status to the server. Which specific mechanism can be used for this is outside this document's scope; see also Section 9.

#### 5.9.2.3. 4.02 Bad Option

The request could not be understood by the server due to one or more unrecognized or malformed critical options. The client SHOULD NOT repeat the request without modification.

#### 5.9.2.4. 4.03 Forbidden

Like HTTP 403 "Forbidden".

#### 5.9.2.5. 4.04 Not Found

Like HTTP 404 "Not Found".

#### 5.9.2.6. 4.05 Method Not Allowed

Like HTTP 405 "Method Not Allowed", but with no parallel to the "Allow" header field.

#### 5.9.2.7. 4.06 Not Acceptable

Like HTTP 406 "Not Acceptable", but with no response entity.

#### 5.9.2.8. 4.12 Precondition Failed

Like HTTP 412 "Precondition Failed".

#### 5.9.2.9. 4.13 Request Entity Too Large

Like HTTP 413 "Request Entity Too Large".

## 5.9.2.10.    4.15 Unsupported Media Type

Like HTTP 415 "Unsupported Media Type".

## 5.9.3.    Server Error 5.xx

This class of response code indicates cases in which the server is aware that it has erred or is incapable of performing the request. These response codes are applicable to any request method.

The server SHOULD include a diagnostic message as detailed in Section 5.5.2.

Responses of this class are cacheable: Caches can use the Max-Age Option to determine freshness (see Section 5.6.1). They cannot be validated.

## 5.9.3.1.    5.00 Internal Server Error

Like HTTP 500 "Internal Server Error".

## 5.9.3.2.    5.01 Not Implemented

Like HTTP 501 "Not Implemented".

## 5.9.3.3.    5.02 Bad Gateway

Like HTTP 502 "Bad Gateway".

## 5.9.3.4.    5.03 Service Unavailable

Like HTTP 503 "Service Unavailable", but using the Max-Age Option in place of the "Retry-After" header field.

## 5.9.3.5.    5.04 Gateway Timeout

Like HTTP 504 "Gateway Timeout".

## 5.9.3.6.    5.05 Proxying Not Supported

The server is unable or unwilling to act as a proxy for the URI specified in the Proxy-Uri Option (see Section 5.10.3).

## 5.10.    Option Definitions

The individual CoAP options are summarized in Table 1 and explained below.



| No. | C | R | Name           | Format | Length  | Default     |
|-----|---|---|----------------|--------|---------|-------------|
| 1   | x |   | Content-Type   | uint   | 0-2 B   | (none)      |
| 2   |   |   | Max-Age        | uint   | 0-4 B   | 60          |
| 3   | x | x | Proxy-Uri      | string | 1-270 B | (none)      |
| 4   |   | x | ETag           | opaque | 1-8 B   | (none)      |
| 5   | x |   | Uri-Host       | string | 1-270 B | (see below) |
| 6   |   | x | Location-Path  | string | 0-270 B | (none)      |
| 7   | x |   | Uri-Port       | uint   | 0-2 B   | (see below) |
| 8   |   | x | Location-Query | string | 0-270 B | (none)      |
| 9   | x | x | Uri-Path       | string | 0-270 B | (none)      |
| 11  | x |   | Token          | opaque | 1-8 B   | (empty)     |
| 12  |   | x | Accept         | uint   | 0-2 B   | (none)      |
| 13  | x | x | If-Match       | opaque | 0-8 B   | (none)      |
| 15  | x | x | Uri-Query      | string | 0-270 B | (none)      |
| 21  | x |   | If-None-Match  | empty  | 0 B     | (none)      |

C=Critical, R=Repeatable

Table 1: Options

#### 5.10.1. Token

The Token Option is used to match a response with a request. Every request has a client-generated token which the server **MUST** echo in any response. A default value of a zero-length token is assumed in the absence of the option. Thus when the token value is empty, the Token Option **SHOULD** be elided for efficiency.

A token is intended for use as a client-local identifier for differentiating between concurrent requests (see Section 5.3). A client **SHOULD** generate tokens in a way that tokens currently in use for a given source/destination pair are unique. An empty token value is appropriate e.g. when no other tokens are in use to a destination, or when requests are made serially per destination. There are however multiple possible implementation strategies to fulfill this. An endpoint receiving a token **MUST** treat it as opaque and make no assumptions about its format.

#### 5.10.2. Uri-Host, Uri-Port, Uri-Path and Uri-Query

The Uri-Host, Uri-Port, Uri-Path and Uri-Query Options are used to specify the target resource of a request to a CoAP origin server. The options encode the different components of the request URI in a way that no percent-encoding is visible in the option values and that the full URI can be reconstructed at any involved endpoint. The

syntax of CoAP URIs is defined in Section 6.

The steps for parsing URIs into options is defined in Section 6.4. These steps result in zero or more Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in a request, where each option holds the following values:

- o the Uri-Host Option specifies the Internet host of the resource being requested,
- o the Uri-Port Option specifies the transport layer port number of the resource,
- o each Uri-Path Option specifies one segment of the absolute path to the resource, and
- o each Uri-Query Option specifies one argument parameterizing the resource.

Note: Fragments ([RFC3986], Section 3.5) are not part of the request URI and thus will not be transmitted in a CoAP request.

The default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. Likewise, the default value of the Uri-Port Option is the destination UDP port. The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers. Explicit Uri-Host and Uri-Port Options are typically used when an endpoint hosts multiple virtual servers.

The Uri-Path and Uri-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Uri-Path Option MUST NOT be "." or ".." (as the request URI must be resolved before parsing it into options).

The steps for constructing the request URI from the options are defined in Section 6.5. Note that an implementation does not necessarily have to construct the URI; it can simply look up the target resource by looking at the individual options.

Examples can be found in Appendix B.

#### 5.10.3. Proxy-Uri

The Proxy-Uri Option is used to make a request to a proxy (see Section 5.7). The proxy is requested to forward the request or service it from a valid cache, and return the response.

The option value is an absolute-URI ([RFC3986], Section 4.3). In case the absolute-URI doesn't fit within a single option, the Proxy-Uri Option MAY be included multiple times in a request such that the concatenation of the values results in the single absolute-URI.

All but the last instance of the Proxy-Uri Option MUST have a value with a length of 270 bytes, and the last instance MUST NOT be empty.

Note that the proxy MAY forward the request on to another proxy or directly to the server specified by the absolute-URI. In order to avoid request loops, a proxy MUST be able to recognize all of its server names, including any aliases, local variations, and the numeric IP addresses.

An endpoint receiving a request with a Proxy-Uri Option that is unable or unwilling to act as a proxy for the request MUST cause the return of a 5.05 (Proxying Not Supported) response.

The Proxy-Uri Option MUST take precedence over any of the Uri-Host, Uri-Port, Uri-Path or Uri-Query options (which MUST NOT be included at the same time).

#### 5.10.4. Content-Type

The Content-Type Option indicates the representation format of the message payload. The representation format is given as a numeric media type identifier that is defined in the CoAP Media Type registry (Section 12.3). No default value is assumed in the absence of the option.

#### 5.10.5. Accept

The CoAP Accept option indicates when included one or more times in a request, one or more media types, each of which is an acceptable media type for the client, in the order of preference (most preferred first). The representation format is given as a numeric media type identifier that is defined in the CoAP Media Type registry (Section 12.3). If no Accept options are given, the client does not express a preference (thus no default value is assumed). The client prefers the representation returned by the server to be in one of the media types indicated. The server SHOULD return one of the preferred media types if available. If none of the preferred media types can be returned, then a 4.06 "Not Acceptable" SHOULD be sent as a response.

Note that as a server might not support the Accept option (and thus would ignore it as it is elective), the client needs to be prepared to receive a representation in a different media type. The client

can simply discard a representation it can not make use of.

#### 5.10.6. Max-Age

The Max-Age Option indicates the maximum time a response may be cached before it **MUST** be considered not fresh (see Section 5.6.1).

The option value is an integer number of seconds between 0 and  $2^{32}-1$  inclusive (about 136.1 years). A default value of 60 seconds is assumed in the absence of the option in a response.

#### 5.10.7. ETag

The ETag Option in a response provides the current value of the entity-tag for the enclosed representation of the target resource.

An entity-tag is intended for use as a resource-local identifier for differentiating between representations of the same resource that vary over time. It may be generated in any number of ways including a version, checksum, hash or time. An endpoint receiving an entity-tag **MUST** treat it as opaque and make no assumptions about its format. (Endpoints generating an entity-tag are encouraged to use the most compact representation possible, in particular in regards to clients and intermediaries that may want to store multiple ETag values.)

An endpoint that has one or more representations previously obtained from the resource can specify the ETag Option in a request for each stored response to determine if any of those representations is current (see Section 5.6.2).

The ETag Option **MUST NOT** occur more than once in a response, and **MAY** occur one or more times in a request.

#### 5.10.8. Location-Path and Location-Query

The Location-Path and Location-Query Options together indicate a relative URI that consists either of an absolute path, a query string or both. A combination of these options is included in a 2.01 (Created) response to indicate the location of the a resource created as the result of a POST request (see Section 5.8.2). The location is resolved relative to the request URI.

If a response with one or more Location-Path and/or Location-Query Options passes through a cache and the implied URI identifies one or more currently stored responses, those entries **SHOULD** be marked as not fresh.

Each Location-Path Option specifies one segment of the absolute path

to the resource, and each Uri-Location Option specifies one argument parameterizing the resource. The Location-Path and Location-Query Option can contain any character sequence. No percent-encoding is performed. The value of a Location-Path Option MUST NOT be "." or "..".

The steps for constructing the location URI from the options are analogous to Section 6.5, except that the first five steps are skipped and the result is a relative URI-reference.

More Location-\* options may be defined in the future, and have been reserved option numbers 44, 46 and 48. If any of these reserved option numbers occurs in addition to Location-Path and/or Location-Query and are not supported, then a 4.02 (Bad Option) error MUST be returned.

#### 5.10.9. If-Match

The If-Match Option MAY be used to make a request conditional on the current existence or value of an ETag for one or more representations of the target resource. If-Match is generally useful for resource update requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource (i.e., the "lost update" problem).

The value of an If-Match option is either an ETag or the empty string. An empty string places the precondition on the existence of any current representation for the target resource.

The If-Match Option can occur multiple times. If any of the ETags given as an option value match the ETag of the current representation for the target resource, or if an If-Match Option with an empty string as option value is given and any current representation exists for the target resource, then the server MAY perform the request method as if the If-Match Option was not present.

If none of the ETags match and, if an empty string is given, no current representation exists at all, the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

If the request would, without the If-Match Options, result in anything other than a 2.xx or 4.12 response code, then any If-Match Options MUST be ignored.

#### 5.10.10. If-None-Match

The If-None-Match Option MAY be used to make a request conditional on the non-existence of the target resource. If-None-Match is useful for resource creation requests, such as PUT requests, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource. The If-None-Match Option carries no value.

If the target resource does exist, then the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

### 6. CoAP URIs

CoAP uses the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. Resources are organized hierarchically and governed by a potential CoAP origin server listening for CoAP requests ("coap") or DTLS-secured CoAP requests ("coaps") on a given UDP port. The CoAP server is identified via the generic syntax's authority component, which includes a host identifier and optional UDP port number. The remainder of the URI is considered to be identifying a resource which can be operated on by the methods defined by the CoAP protocol. The "coap" and "coaps" URI schemes can thus be compared to the "http" and "https" URI schemes respectively.

The syntax of the "coap" and "coaps" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", "query", "segment", "IP-literal", "IPv4address" and "reg-name" are adopted from [RFC3986].

#### 6.1. coap URI Scheme

```
coap-URI = "coap:" "://" host [ ":" port ] path-abempty [ "?" query ]
```

If host is provided as an IP-literal or IPv4address, then the CoAP server can be reached at that IP address. If host is a registered name, then that name is considered an indirect identifier and the endpoint might use a name resolution service, such as DNS, to find the address of that host. The host MUST NOT be empty. The port subcomponent indicates the UDP port at which the CoAP server is located. If it is empty or not given, then the default port 5683 is assumed.

The path identifies a resource within the scope of the host and port. It consists of a sequence of path segments separated by a slash

character (U+002F SOLIDUS "/").

The query serves to further parameterize the resource. It consists of a sequence of arguments separated by an ampersand character (U+0026 AMPERSAND "&"). An argument is often in the form of a "key=value" pair.

The "coap" URI scheme supports the path prefix "/.well-known/" defined by [RFC5785] for "well-known locations" in the name-space of a host. This enables discovery of policy or other information about a host ("site-wide metadata"), such as hosted resources (see Section 7).

Application designers are encouraged to make use of short, but descriptive URIs. As the environments that CoAP is used in are usually constrained for bandwidth and energy, the trade-off between these two qualities should lean towards the shortness, without ignoring descriptiveness.

## 6.2. coaps URI Scheme

```
coaps-URI = "coaps:" "/" host [ ":" port ] path-abempty
           [ "?" query ]
```

All of the requirements listed above for the "coap" scheme are also requirements for the "coaps" scheme, except that a default UDP port of [IANA\_TBD\_PORT] is assumed if the port subcomponent is empty or not given, and the UDP datagrams MUST be secured for privacy through the use of DTLS as described in Section 9.1.

Unlike the "coap" scheme, responses to "coaps" identified requests are never "public" and thus MUST NOT be reused for shared caching. They can, however, be reused in a private cache if the message is cacheable by default in CoAP.

Resources made available via the "coaps" scheme have no shared identity with the "coap" scheme even if their resource identifiers indicate the same authority (the same host listening to the same UDP port). They are distinct name spaces and are considered to be distinct origin servers.

## 6.3. Normalization and Comparison Rules

Since the "coap" and "coaps" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in [RFC3986], Section 6, using the defaults described above for each scheme.

If the port is equal to the default port for a scheme, the normal form is to elide the port subcomponent. Likewise, an empty path component is equivalent to an absolute path of "/", so the normal form is to provide a path of "/" instead. The scheme and host are case-insensitive and normally provided in lowercase; IP-literals are in recommended form [RFC5952]; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded octets (see [RFC3986], Section 2.1): the normal form is to not encode them.

For example, the following three URIs are equivalent, and cause the same options and option values to appear in the CoAP messages:

```
coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Esensors/temp.xml
coap://EXAMPLE.com:/%7esensors/temp.xml
```

#### 6.4. Decomposing URIs into Options

The steps to parse a request's options from a string `/url/` are as follows. These steps either result in zero or more of the Uri-Host, Uri-Port, Uri-Path and Uri-Query Options being included in the request, or they fail.

1. If the `/url/` string is not an absolute URI ([RFC3986]), then fail this algorithm.
2. Resolve the `/url/` string using the process of reference resolution defined by [RFC3986], with the URL character encoding set to UTF-8 [RFC3629].

NOTE: It doesn't matter what it is resolved relative to, since we already know it is an absolute URL at this point.

3. If `/url/` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.
4. If `/url/` has a `<fragment>` component, then fail this algorithm.
5. If the `<host>` component of `/url/` does not represent the request's destination IP address as an IP-literal or IPv4address, include a Uri-Host Option and let that option's value be the value of the `<host>` component of `/url/`, converted to ASCII lowercase, and then converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

NOTE: In the usual case where the request's destination IP



address is derived from the host part, this ensures that Uri-Host Options are only used for host parts of the form reg-name.

6. If /url/ has a <port> component, then let /port/ be that component's value interpreted as a decimal integer; otherwise, let /port/ be the default port for the scheme.
7. If /port/ does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be /port/.
8. If the value of the <path> component of /url/ is empty or consists of a single slash character (U+002F SOLIDUS "/"), then move to the next step.

Otherwise, for each segment in the <path> component, include a Uri-Path Option and let that option's value be the segment (not including the delimiting slash characters) after converting all percent-encodings ("% followed by two hexadecimal digits) to the corresponding characters.

9. If /url/ has a <query> component, then, for each argument in the <query> component, include a Uri-Query Option and let that option's value be the argument (not including the question mark and the delimiting ampersand characters) after converting all percent-encodings to the corresponding characters.

Note that these rules completely resolve any percent-encoding.

#### 6.5. Composing URIs from Options

The steps to construct a URI from a request's options are as follows. These steps either result in a URI, or they fail. In these steps, percent-encoding a character means replacing each of its (UTF-8 encoded) bytes by a "%" character followed by two hexadecimal digits representing the byte, where the digits A-F are in upper case (as defined in [RFC3986] Section 2.1; to reduce variability, the hexadecimal notation for percent-encoding in CoAP URIs MUST use uppercase letters). The definitions of "unreserved" and "sub-delims" are adopted from [RFC3986].

1. If the request is secured using DTLS, let /url/ be the string "coaps://". Otherwise, let /url/ be the string "coap://".
2. If the request includes a Uri-Host Option, let /host/ be that option's value, where any non-ASCII characters are replaced by their corresponding percent-encoding. If /host/ is not a valid reg-name or IP-literal or IPv4address, fail the algorithm. Otherwise, let /host/ be the IP-literal (making use of the

conventions of [RFC5952]) or IPv4address representing the request's destination IP address.

3. Append /host/ to /url/.
4. If the request includes a Uri-Port Option, let /port/ be that option's value. Otherwise, let /port/ be the request's destination UDP port.
5. If /port/ is not the default port for the scheme, then append a single U+003A COLON character (:) followed by the decimal representation of /port/ to /url/.
6. Let /resource name/ be the empty string. For each Uri-Path Option in the request, append a single character U+002F SOLIDUS (/) followed by the option's value to /resource name/, after converting any character that is not either in the "unreserved" set, "sub-delims" set, a U+003A COLON (:) or U+0040 COMMERCIAL AT (@) character, to its percent-encoded form.
7. If /resource name/ is the empty string, set it to a single character U+002F SOLIDUS (/).
8. For each Uri-Query Option in the request, append a single character U+003F QUESTION MARK (?) (first option) or U+0026 AMPERSAND (&) (subsequent options) followed by the option's value to /resource name/, after converting any character that is not either in the "unreserved" set, "sub-delims" set (except U+0026 AMPERSAND (&)), a U+003A COLON (:), U+0040 COMMERCIAL AT (@), U+002F SOLIDUS (/) or U+003F QUESTION MARK (?) character, to its percent-encoded form.
9. Append /resource name/ to /url/.
10. Return /url/.

Note that these steps have been designed to lead to a URI in normal form (see Section 6.3).

## 7. Discovery

### 7.1. Service Discovery

A server is discovered by a client by the client knowing or learning a URI that references a resource in the namespace of the server. Alternatively, clients can use Multicast CoAP (see Section 8) and the "All CoAP Nodes" multicast address to find CoAP servers.

Unless the port subcomponent in a "coap" or "coaps" URI indicates the UDP port at which the CoAP server is located, the server is assumed to be reachable at the default port.

The CoAP default port number 5683 MUST be supported by a server for resource discovery (see Section 7.2 below) and SHOULD be supported for providing access to other resources. The default port number [IANA\_TBD\_PORT] for DTLS-secured CoAP MAY be supported by a server for resource discovery and for providing access to other resources. In addition other endpoints may be hosted in the dynamic port space.

When a CoAP server is hosted by a 6LoWPAN node, it SHOULD also support a port number in the 61616-61631 compressed UDP port space defined in [RFC4944] (note that, as its UDP port differs from the default port, it is a different endpoint from the server at the default port). So if the default port number does not work and a client knows that the CoAP server is hosted by a 6LoWPAN node, the client MAY try to contact the CoAP server at a port number in the 61616-61631 space.

## 7.2. Resource Discovery

The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. A CoAP endpoint SHOULD support the CoRE Link Format of discoverable resources as described in [I-D.ietf-core-link-format]. It is up to the server which resources are made discoverable (if any).

### 7.2.1. 'ct' Attribute

This section defines a new Web Linking [RFC5988] attribute for use with [I-D.ietf-core-link-format]. The Content-type code "ct" attribute provides a hint about the Internet media type(s) this resource returns. Note that this is only a hint, and does not override the Content-type Option of a CoAP response obtained by actually following the link. The value is in the CoAP identifier code format as a decimal ASCII integer and MUST be in the range of 0-65535 (16-bit unsigned integer). For example application/xml would be indicated as "ct=41". If no Content-type code attribute is present then nothing about the type can be assumed. The Content-type code attribute MAY appear more than once in a link, indicating that multiple content-types are available.

```
link-extension    = <Defined in RFC5988>
link-extension    = ( "ct" "=" cardinal ) ; Range of 0-65535
cardinal          = "0" / %x31-39 *DIGIT
```

## 8. Multicast CoAP

CoAP supports making requests to a IP multicast group. This is defined by a series of deltas to Unicast CoAP.

### 8.1. Messaging Layer

A multicast request is characterized by being transported in a CoAP message that is addressed to an IP multicast address instead of a CoAP end-point. Such multicast requests **MUST** be Non-Confirmable.

Some mechanisms for avoiding congestion from multicast requests have been considered in [I-D.eggert-core-congestion-control].

A server **SHOULD** be aware that a request arrived via multicast, e.g. by making use of modern APIs such as IPV6\_RECVPKTINFO [RFC3542], if available.

When a server is aware that a request arrived via multicast, it **MUST NOT** return a RST in reply to NON. If it is not aware, it **MAY** return a RST in reply to NON as usual.

### 8.2. Request/Response Layer

When a server is aware that a request arrived via multicast, the server **MAY** always pretend it did not receive the request, in particular if it doesn't have anything useful to respond (e.g., if it only has an empty payload or an error response). The decision for this may depend on the application. (For example, in [I-D.ietf-core-link-format] query filtering, a server should not respond to a multicast request if the filter does not match.)

If a server does decide to respond to a multicast request, it should not respond immediately. Instead, it should pick a duration for the period of time during which it intends to respond. For purposes of this exposition, we call the length of this period the *Leisure*. The specific value of this *Leisure* may depend on the application, or **MAY** be derived as described below. The server **SHOULD** then pick a random point of time within the chosen *Leisure* period to send back the unicast response to the multicast request.

To compute a value for *Leisure*, the server should have a group size estimate *G*, a target rate *R* (which both should be chosen conservatively) and an estimated response size *S*; a rough lower bound for *Leisure* can then be computed as

$$lb\_Leisure = S * G / R$$

E.g., for a multicast request with link-local scope on an 2.4 GHz

IEEE 802.15.4 (6LoWPAN) network, G could be (relatively conservatively) set to 100, S to 100 bytes, and the target rate to a conservative 8 kbit/s = 1 kB/s. The resulting lower bound for the Leisure is 10 seconds.

When matching a response to a multicast request, only the token **MUST** match; the source endpoint of the response does not need to (and will not) be the same as the destination endpoint of the original request.

#### 8.2.1. Caching

When a client makes a multicast request, it always makes a new request to the multicast group (since there may be new group members that joined meanwhile or ones that did not get the previous request). It **MAY** update the cache with the received responses. Then it uses both cached-still-fresh and 'new' responses as the result of the request.

A response received in reply to a GET request to a multicast group **MAY** be used to satisfy a subsequent request on the related unicast request URI. The unicast request URI is obtained by replacing the authority part of the request URI with the transport layer source address of the response message.

A cache **MAY** revalidate a response by making a GET request on the related unicast request URI.

A GET request to a multicast group **MUST NOT** contain an ETag option. A mechanism to suppress responses the client already has is left for further study.

#### 8.2.2. Proxying

When a forward proxy receives a request with a Proxy-Uri that indicates a multicast address, the proxy obtains a set of responses as described above and sends all responses (both cached-still-fresh and new) back to the original client.

### 9. Securing CoAP

This section defines the DTLS binding for CoAP, and the alternative use of IPsec.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials and access control lists. This specification defines provisioning for the RawPublicKey mode in Section 9.1.3.2.1. At the end of the

provisioning phase, the device will be in one of four security modes with the following information for the given mode. The NoSec and RawPublicKey modes are mandatory to implement for this specification.

NoSec: There is no protocol level security (DTLS is disabled).

Alternative techniques to provide lower layer security SHOULD be used when appropriate. The use of IPsec is discussed in Section 9.2.

PreSharedKey: DTLS is enabled and there is a list of pre-shared keys [RFC4279] and each key includes a list of which nodes it can be used to communicate with as described in Section 9.1.3.1. At the extreme there may be one key for each node this CoAP node needs to communicate with (1:1 node/key ratio).

RawPublicKey: DTLS is enabled and the device has a raw public key certificate that is validated using an out-of-band mechanism [I-D.ietf-tls-oob-pubkey] as described in Section 9.1.3.2. The device also has an identity calculated from the public key and a list of identities of the nodes it can communicate with.

Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 certificate [RFC5280] that binds it to its Authority Name and is signed by some common trust root as described in Section 9.1.3.3. The device also has a list of root trust anchors that can be used for validating a certificate.

In the "NoSec" mode, the system simply sends the packets over normal UDP over IP and is indicated by the "coap" scheme and the CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes; see Section 11.5 for an additional complication with this approach.

The other three security modes are achieved using DTLS and are indicated by the "coaps" scheme and DTLS-secured CoAP default port. The result is a security association that can be used to authenticate (within the limits of the security model) and, based on this authentication, authorize the communication partner. CoAP itself does not provide protocol primitives for authentication or authorization; where this is required, it can either be provided by communication security (i.e., IPsec or DTLS) or by object security (within the payload). Devices that require authorization for certain operations are expected to require one of these two forms of security. Necessarily, where an intermediary is involved, communication security only works when that intermediary is part of the trust relationships; CoAP does not provide a way to forward different levels of authorization that clients may have with an

intermediary to further intermediaries or origin servers -- it therefore may be required to perform all authorization at the first intermediary.

### 9.1. DTLS-secured CoAP

Just as HTTP is secured using Transport Layer Security (TLS) over TCP, CoAP is secured using Datagram TLS (DTLS) [RFC6347] over UDP (see Figure 10). This section defines the CoAP binding to DTLS, along with the minimal mandatory-to-implement configurations appropriate for constrained environments. The binding is defined by a series of deltas to Unicast CoAP. DTLS is in practice TLS with added features to deal with the unreliable nature of the UDP transport.

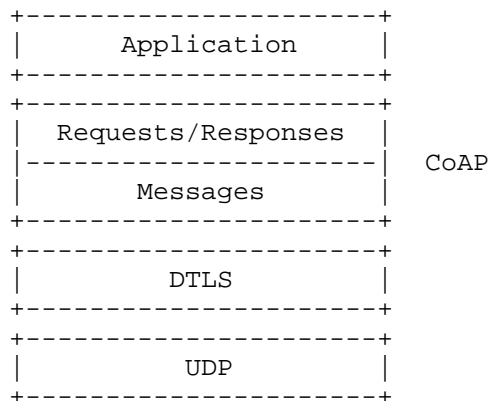


Figure 10: Abstract layering of DTLS-secured CoAP

In some constrained nodes (limited flash and/or RAM) and networks (limited bandwidth or high scalability requirements), and depending on the specific cipher suites in use, DTLS may not be applicable. Some of DTLS' cipher suites can add significant implementation complexity as well as some initial handshake overhead needed when setting up the security association. Once the initial handshake is completed, DTLS adds a limited per-datagram overhead of approximately 13 bytes, not including any initialization vectors/nonces (e.g., 8 bytes with TLS\_PSK\_WITH\_AES\_128\_CCM\_8 [I-D.mcgre-w-tls-aes-ccm]), integrity check values (e.g., 8 bytes with TLS\_PSK\_WITH\_AES\_128\_CCM\_8 [I-D.mcgre-w-tls-aes-ccm]) and padding required by the cipher suite. Whether and which mode of using DTLS is applicable for a CoAP-based application should be carefully weighed considering the specific cipher suites that may be applicable, and whether the session maintenance makes it compatible with application flows and sufficient resources are available on the constrained nodes and for the added

network overhead. DTLS is not applicable to group keying (multicast communication); however, it may be a component in a future group key management protocol.

#### 9.1.1.1. Messaging Layer

The endpoint acting as the CoAP client should also act as the DTLS client. It should initiate a session to the server on the appropriate port. When the DTLS handshake has finished, the client may initiate the first CoAP request. All CoAP messages MUST be sent as DTLS "application data".

The following rules are added for matching an ACK or RST to a CON message or a RST to a NON message are as follows: The DTLS session MUST be the same and the epoch MUST be the same.

A message is the same when it is sent within the same DTLS session and same epoch and has the same Message ID.

Note: When a confirmable message is retransmitted, a new DTLS sequence\_number is used for each attempt, even though the CoAP Message ID stays the same. So a recipient still has to perform deduplication as described in Section 4.5. Retransmissions MUST NOT be performed across epochs.

DTLS connections in RawPublicKey and Certificate mode are set up using mutual authentication so they can remain up and be reused for future message exchanges in either direction. Devices can close a DTLS connection when they need to recover resources but in general they should keep the connection up for as long as possible. Closing the DTLS connection after every CoAP message exchange is very inefficient.

#### 9.1.1.2. Request/Response Layer

The following rules are added for matching a response to a request: The DTLS session MUST be the same and the epoch MUST be the same.

##### 9.1.2.1. Caching

The following rules are added for using a response that was obtained using DTLS-secured CoAP: For a presented request, a CoAP endpoint MUST NOT use a stored response, unless the identity is the same.

##### 9.1.2.2. Proxying

Responses to "coaps" identified requests are never "public" and thus MUST NOT be reused for shared caching. They can, however, be reused



in a private cache if the message is cacheable by default in CoAP.

#### 9.1.3. Endpoint Identity

Devices SHOULD support the Server Name Indication (SNI) to indicate their Authority Name in the SNI HostName field as defined in Section 3 of [RFC6066]. This is needed so that when a host that acts as a virtual server for multiple Authorities receives a new DTLS connection, it knows which keys to use for the DTLS session.

##### 9.1.3.1. Pre-Shared Keys

When forming a connection to a new node, the system selects an appropriate key based on which nodes it is trying to reach and then forms a DTLS session using a PSK (Pre-Shared Key) mode of DTLS. Implementations in these modes MUST support the mandatory to implement cipher suite TLS\_PSK\_WITH\_AES\_128\_CCM\_8 as specified in [I-D.mcgregw-tls-aes-ccm].

The security considerations of [RFC4279] (Section 7) apply. In particular, applications should carefully weigh whether they need Perfect Forward Secrecy (PFS) or not and select an appropriate cipher suite (7.1). The entropy of the PSK must be sufficient to mitigate against brute-force and (where the PSK is not chosen randomly but by a human) dictionary attacks (7.2). The cleartext communication of client identities may leak data or compromise privacy (7.3).

##### 9.1.3.2. Raw Public Key Certificates

In this mode the device has an asymmetric key pair but without an X.509 certificate (called a raw public key). A device MAY be configured with multiple raw public keys. The type and length of the raw public key depends on the cipher suite used. Implementations in RawPublicKey mode MUST support the mandatory to implement cipher suite TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CCM\_8 as specified in [I-D.mcgregw-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. The mechanism for using raw public keys with TLS is specified in [I-D.ietf-tls-oob-pubkey].

##### 9.1.3.2.1. Provisioning

The RawPublicKey mode was designed to be easily provisioned in M2M deployments. It is assumed that each device has an appropriate asymmetric public key pair installed. An identifier is calculated from the public key as described in Section 2 of [I-D.farrell-decade-ni]. All implementations that support checking RawPublicKey identities MUST support at least the sha-256-120 mode (SHA-256 truncated to 120 bits). Implementations SHOULD support also

longer length identifiers and MAY support shorter lengths. Note that the shorter lengths provide less security against attacks and their use is NOT RECOMMENDED.

Depending on how identifiers are given to the system that verifies them, support for URI, binary, and/or human-speakable format [I-D.farrell-decade-ni] needs to be implemented. All implementations SHOULD support the binary mode and implementations that have a user interface SHOULD also support the human-speakable format.

During provisioning, the identifier of each node is collected, for example by reading a barcode on the outside of the device or by obtaining a pre-compiled list of the identifiers. These identifiers are then installed in the corresponding endpoint, for example an M2M data collection server. The identifier is used for two purposes, to associate the endpoint with further device information and to perform access control. During provisioning, an access control list of identifiers the device may start DTLS sessions with SHOULD also be installed.

#### 9.1.3.3. X.509 Certificates

Implementations in Certificate Mode MUST support the mandatory to implement cipher suite TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CCM\_8 as specified in [RFC5246].

The Authority Name in the certificate is the name that would be used in the Authority part of a CoAP URI. It is worth noting that this would typically not be either an IP address or DNS name but would instead be a long term unique identifier for the device such as the EUI-64 [EUI64]. The discovery process used in the system would build up the mapping between IP addresses of the given devices and the Authority Name for each device. Some devices could have more than one Authority and would need more than a single certificate.

When a new connection is formed, the certificate from the remote device needs to be verified. If the CoAP node has a source of absolute time, then the node SHOULD check the validity dates of the certificate are within range. The certificate MUST also be signed by an appropriate chain of trust. If the certificate contains a SubjectAltName, then the Authority Name MUST match at least one of the authority names of any CoAP URI found in a URI type fields in the SubjectAltName set. If there is no SubjectAltName in the certificate, then the Authoritative Name must match the CN found in the certificate using the matching rules defined in [RFC2818] with the exception that certificates with wildcards are not allowed.

If the system has a shared key in addition to the certificate, then a

cipher suite that includes the shared key such as  
TLS\_RSA\_PSK\_WITH\_AES\_128\_CBC\_SHA [RFC4279] SHOULD be used.

## 9.2. Using CoAP with IPsec

One mechanism to secure CoAP in constrained environments is the IPsec Encapsulating Security Payload (ESP) [RFC4303] when CoAP is used without DTLS in NoSec Mode. Using IPsec ESP with the appropriate configuration, it is possible for many constrained devices to support encryption with built-in link-layer encryption hardware. For example, some IEEE 802.15.4 radio chips are compatible with AES-CBC (with 128-bit keys) [RFC3602] as defined for use with IPsec in [RFC4835]. Alternatively, particularly on more common IEEE 802.15.4 hardware that supports AES encryption but not decryption, and to avoid the need for padding, nodes could directly use the more widely supported AES-CCM as defined for use with IPsec in [RFC4309], if the security considerations in Section 9 of that specification can be fulfilled.

Necessarily for AES-CCM, but much preferably also for AES-CBC, static keying should be avoided and the initial keying material be derived into transient session keys, e.g. using a low-overhead mode of IKEv2 [RFC5996] as described in [I-D.kivinen-ipsecme-ikev2-minimal]; such a protocol for managing keys and sequence numbers is also the only way to achieve anti-replay capabilities. However, no recommendation can be made at this point on how to manage group keys (i.e., for multicast) in a constrained environment. Once any initial setup is completed, IPsec ESP adds a limited overhead of approximately 10 bytes per packet, not including initialization vectors, integrity check values and padding required by the cipher suite.

When using IPsec to secure CoAP, both authentication and confidentiality SHOULD be applied as recommended in [RFC4303]. The use of IPsec between CoAP endpoints is transparent to the application layer and does not require special consideration for a CoAP implementation.

IPsec may not be appropriate for all environments. For example, IPsec support is not available for many embedded IP stacks and even in full PC operating systems or on back-end web servers, application developers may not have sufficient access to configure or enable IPsec or to add a security gateway to the infrastructure. Problems with firewalls and NATs may furthermore limit the use of IPsec.

## 10. Cross-Protocol Proxying between CoAP and HTTP

CoAP supports a limited subset of HTTP functionality, and thus cross-

protocol proxying to HTTP is straightforward. There might be several reasons for proxying between CoAP and HTTP, for example when designing a web interface for use over either protocol or when realizing a CoAP-HTTP proxy. Likewise, CoAP could equally be proxied to other protocols such as XMPP [RFC6120] or SIP [RFC3264]; the definition of these mechanisms is out of scope of this specification.

There are two possible directions to access a resource via a forward proxy:

**CoAP-HTTP Proxying:** Enables CoAP clients to access resources on HTTP servers through an intermediary. This is initiated by including the Proxy-Uri Option with an "http" or "https" URI in a CoAP request to a CoAP-HTTP proxy.

**HTTP-CoAP Proxying:** Enables HTTP clients to access resources on CoAP servers through an intermediary. This is initiated by specifying a "coap" or "coaps" URI in the Request-Line of an HTTP request to an HTTP-CoAP proxy.

Either way, only the Request/Response model of CoAP is mapped to HTTP. The underlying model of confirmable or non-confirmable messages, etc., is invisible and MUST have no effect on a proxy function. The following sections describe the handling of requests to a forward proxy. Reverse proxies are not specified as the proxy function is transparent to the client with the proxy acting as if it was the origin server.

#### 10.1. CoAP-HTTP Mapping

If a request contains a Proxy-URI Option with an 'http' or 'https' URI [RFC2616], then the receiving CoAP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated HTTP resource and return the result to the client.

This section specifies for any CoAP request the CoAP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to an HTTP origin server.

Since HTTP and CoAP share the basic set of request methods, performing a CoAP request on an HTTP resource is not so different from performing it on a CoAP resource. The meanings of the individual CoAP methods when performed on HTTP resources are explained below.

If the proxy is unable or unwilling to service a request with an HTTP URI, a 5.05 (Proxying Not Supported) response SHOULD be returned to the client. If the proxy services the request by interacting with a third party (such as the HTTP origin server) and is unable to obtain a result within a reasonable time frame, a 5.04 (Gateway Timeout) response SHOULD be returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response SHOULD be returned.

#### 10.1.1. GET

The GET method requests the proxy to return a representation of the HTTP resource identified by the request URI.

Upon success, a 2.05 (Content) response SHOULD be returned. The payload of the response MUST be a representation of the target HTTP resource, and the Content-Type Option be set accordingly. The response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the HTTP entity has an entity tag, the proxy SHOULD include an ETag Option in the response and process ETag Options in requests as described below.

A client can influence the processing of a GET request by including the following option:

Accept: The request MAY include one or more Accept Options, identifying the preferred response content-type.

ETag: The request MAY include one or more ETag Options, identifying responses that the client has stored. This requests the proxy to send a 2.03 (Valid) response whenever it would send a 2.05 (Content) response with an entity tag in the requested set otherwise.

#### 10.1.2. PUT

The PUT method requests the proxy to update or create the HTTP resource identified by the request URI with the enclosed representation.

If a new resource is created at the request URI, a 2.01 (Created) response MUST be returned to the client. If an existing resource is modified, a 2.04 (Changed) response MUST be returned to indicate successful completion of the request.

#### 10.1.1.3. DELETE

The DELETE method requests the proxy to delete the HTTP resource identified by the request URI at the HTTP origin server.

A 2.02 (Deleted) response MUST be returned to client upon success or if the resource does not exist at the time of the request.

#### 10.1.1.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the HTTP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 2.04 (Changed) response MUST be returned to the client. If a resource has been created on the origin server, a 2.01 (Created) response MUST be returned.

### 10.2. HTTP-CoAP Mapping

If an HTTP request contains a Request-URI with a 'coap' or 'coaps' URI, then the receiving HTTP endpoint (called "the proxy" henceforth) is requested to perform the operation specified by the request method on the indicated CoAP resource and return the result to the client.

This section specifies for any HTTP request the HTTP response that the proxy should return to the client. How the proxy actually satisfies the request is an implementation detail, although the typical case is expected to be the proxy translating and forwarding the request to a CoAP origin server. The meanings of the individual HTTP methods when performed on CoAP resources are explained below.

If the proxy is unable or unwilling to service a request with a CoAP URI, a 501 (Not Implemented) response SHOULD be returned to the client. If the proxy services the request by interacting with a third party (such as the CoAP origin server) and is unable to obtain a result within a reasonable time frame, a 504 (Gateway Timeout) response SHOULD be returned; if a result can be obtained but is not understood, a 502 (Bad Gateway) response SHOULD be returned.

#### 10.2.1. OPTIONS and TRACE

As the OPTIONS and TRACE methods are not supported in CoAP a 501 (Not Implemented) error MUST be returned to the client.

### 10.2.2. GET

The GET method requests the proxy to return a representation of the CoAP resource identified by the Request-URI.

Upon success, a 200 (OK) response SHOULD be returned. The payload of the response MUST be a representation of the target CoAP resource, and the Content-Type Option be set accordingly. The response MUST indicate a Max-Age value that is no greater than the remaining time the representation can be considered fresh. If the CoAP entity has an entity tag, the proxy SHOULD include an ETag Option in the response.

A client can influence the processing of a GET request by including the following option:

**Accept:** Each individual Media-type of the HTTP Accept header in a request is mapped to a CoAP Accept option. HTTP Accept Media-type ranges, parameters and extensions are not supported by the CoAP Accept option. If the proxy cannot send a response which is acceptable according to the combined Accept field value, then the proxy SHOULD send a 406 (not acceptable) response.

**Conditional GETs:** Conditional HTTP GET requests that include an "If-Match" or "If-None-Match" request-header field can be mapped to a corresponding CoAP request. The "If-Modified-Since" and "If-Unmodified-Since" request-header fields are not directly supported by CoAP, but SHOULD be implemented locally by a caching proxy.

### 10.2.3. HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response.

Although there is no direct equivalent of HTTP's HEAD method in CoAP, an HTTP-CoAP proxy responds to HEAD requests for CoAP resources, and the HTTP headers are returned without a message-body.

### 10.2.4. POST

The POST method requests the proxy to have the representation enclosed in the request be processed by the CoAP origin server. The actual function performed by the POST method is determined by the origin server and dependent on the resource identified by the request URI.

If the action performed by the POST method does not result in a resource that can be identified by a URI, a 200 (OK) or 204 (No

Content) response MUST be returned to the client. If a resource has been created on the origin server, a 201 (Created) response MUST be returned.

#### 10.2.5. PUT

The PUT method requests the proxy to update or create the CoAP resource identified by the Request-URI with the enclosed representation.

If a new resource is created at the Request-URI, a 201 (Created) response MUST be returned to the client. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.

#### 10.2.6. DELETE

The DELETE method requests the proxy to delete the CoAP resource identified by the Request-URI at the CoAP origin server.

A successful response SHOULD be 200 (OK) if the response includes an entity describing the status or 204 (No Content) if the action has been enacted but the response does not include an entity.

#### 10.2.7. CONNECT

This method can not currently be satisfied by an HTTP-CoAP proxy function as TLS to DTLS tunneling has not been specified. It is however expected that such a tunneling mapping will be defined in the future. A 501 (Not Implemented) error SHOULD be returned to the client.

### 11. Security Considerations

This section analyzes the possible threats to the protocol. It is meant to inform protocol and application developers about the security limitations of CoAP as described in this document. As CoAP realizes a subset of the features in HTTP/1.1, the security considerations in Section 15 of [RFC2616] are also pertinent to CoAP. This section concentrates on describing limitations specific to CoAP.

#### 11.1. Protocol Parsing, Processing URIs

A network-facing application can exhibit vulnerabilities in its processing logic for incoming packets. Complex parsers are well-known as a likely source of such vulnerabilities, such as the ability to remotely crash a node, or even remotely execute arbitrary code on



it. CoAP attempts to narrow the opportunities for introducing such vulnerabilities by reducing parser complexity, by giving the entire range of encodable values a meaning where possible, and by aggressively reducing complexity that is often caused by unnecessary choice between multiple representations that mean the same thing. Much of the URI processing has been moved to the clients, further reducing the opportunities for introducing vulnerabilities into the servers. Even so, the URI processing code in CoAP implementations is likely to be a large source of remaining vulnerabilities and should be implemented with special care. The most complex parser remaining could be the one for the link-format, although this also has been designed with a goal of reduced implementation complexity [I-D.ietf-core-link-format]. (See also section 15.2 of [RFC2616].)

### 11.2. Proxying and Caching

As mentioned in 15.7 of [RFC2616], proxies are by their very nature men-in-the-middle, breaking any IPsec or DTLS protection that a direct CoAP message exchange might have. They are therefore interesting targets for breaking confidentiality or integrity of CoAP message exchanges. As noted in [RFC2616], they are also interesting targets for breaking availability.

The threat to confidentiality and integrity of request/response data is amplified where proxies also cache. Note that CoAP does not define any of the cache-suppressing Cache-Control options that HTTP/1.1 provides to better protect sensitive data.

Finally, a proxy that fans out Separate Responses (as opposed to Piggy-backed Responses) to multiple original requesters may provide additional amplification (see below).

### 11.3. Risk of amplification

CoAP servers generally reply to a request packet with a response packet. This response packet may be significantly larger than the request packet. An attacker might use CoAP nodes to turn a small attack packet into a larger attack packet, an approach known as amplification. There is therefore a danger that CoAP nodes could become implicated in denial of service (DoS) attacks by using the amplifying properties of the protocol: An attacker that is attempting to overload a victim but is limited in the amount of traffic it can generate, can use amplification to generate a larger amount of traffic.

This is particularly a problem in nodes that enable NoSec access, that are accessible from an attacker and can access potential victims (e.g. on the general Internet), as the UDP protocol provides no way

to verify the source address given in the request packet. An attacker need only place the IP address of the victim in the source address of a suitable request packet to generate a larger packet directed at the victim.

As a mitigating factor, many constrained networks will only be able to generate a small amount of traffic, which may make CoAP nodes less attractive for this attack. However, the limited capacity of the constrained network makes the network itself a likely victim of an amplification attack.

A CoAP server can reduce the amount of amplification it provides to an attacker by using slicing/blocking modes of CoAP [I-D.ietf-core-block] and offering large resource representations only in relatively small slices. E.g., for a 1000 byte resource, a 10-byte request might result in an 80-byte response (with a 64-byte block) instead of a 1016-byte response, considerably reducing the amplification provided.

CoAP also supports the use of multicast IP addresses in requests, an important requirement for M2M. Multicast CoAP requests may be the source of accidental or deliberate denial of service attacks, especially over constrained networks. This specification attempts to reduce the amplification effects of multicast requests by limiting when a response is returned. To limit the possibility of malicious use, CoAP servers SHOULD NOT accept multicast requests that can not be authenticated. If possible a CoAP server SHOULD limit the support for multicast requests to specific resources where the feature is required.

On some general purpose operating systems providing a Posix-style API, it is not straightforward to find out whether a packet received was addressed to a multicast address. While many implementations will know whether they have joined a multicast group, this creates a problem for packets addressed to multicast addresses of the form FF0x::1, which are received by every IPv6 node. Implementations SHOULD make use of modern APIs such as IPV6\_RECVPKTINFO [RFC3542], if available, to make this determination.

#### 11.4. IP Address Spoofing Attacks

Due to the lack of a handshake in UDP, a rogue endpoint which is free to read and write messages carried by the constrained network (i.e. NoSec or PreSharedKey deployments with nodes/key ratio > 1:1), may easily attack a single endpoint, a group of endpoints, as well as the whole network e.g. by:

1. spoofing RST in response to a CON or NON message, thus making an endpoint "deaf"; or
2. spoofing the entire response with forged payload/options (this has different levels of impact: from single response disruption, to much bolder attacks on the supporting infrastructure, e.g. poisoning proxy caches, or tricking validation / lookup interfaces in resource directories and, more generally, any component that stores global network state and uses CoAP as the messaging facility to handle state set/update's is a potential target.); or
3. spoofing a multicast request for a target node which may result in both network congestion/collapse and victim DoS'ing / forced wakeup from sleeping; or
4. spoofing observe messages, etc.

In principle, spoofing can be detected by CoAP only in case CON semantics is used, because of unexpected ACK/RSTs coming from the deceived endpoint. But this imposes keeping track of the used Message IDs which is not always possible, and moreover detection becomes available usually after the damage is already done. This kind of attack can be prevented using security modes other than NoSec.

#### 11.5. Cross-Protocol Attacks

The ability to incite a CoAP endpoint to send packets to a fake source address can be used not only for amplification, but also for cross-protocol attacks:

- o the attacker sends a message to a CoAP endpoint with a fake source address,
- o the CoAP endpoint replies with a message to the given source address,
- o the victim at the given source address receives a UDP packet that it interprets according to the rules of a different protocol.

This may be used to circumvent firewall rules that prevent direct communication from the attacker to the victim, but happen to allow communication from the CoAP endpoint (which may also host a valid role in the other protocol) to the victim.

Also, CoAP endpoints may be the victim of a cross-protocol attack generated through an endpoint of another UDP-based protocol such as

DNS. In both cases, attacks are possible if the security properties of the endpoints rely on checking IP addresses (and firewalling off direct attacks sent from outside using fake IP addresses). In general, because of their lack of context, UDP-based protocols are relatively easy targets for cross-protocol attacks.

Finally, CoAP URIs transported by other means could be used to incite clients to send messages to endpoints of other protocols.

One mitigation against cross-protocol attacks is strict checking of the syntax of packets received, combined with sufficient difference in syntax. As an example, it might help if it were difficult to incite a DNS server to send a DNS response that would pass the checks of a CoAP endpoint. Unfortunately, the first two bytes of a DNS reply are an ID that can be chosen by the attacker, which map into the interesting part of the CoAP header, and the next two bytes are then interpreted as CoAP's Message ID (i.e., any value is acceptable). The DNS count words may be interpreted as multiple instances of a (non-existent, but elective) CoAP option 0. The echoed query finally may be manufactured by the attacker to achieve a desired effect on the CoAP endpoint; the response added by the server (if any) might then just be interpreted as added payload.

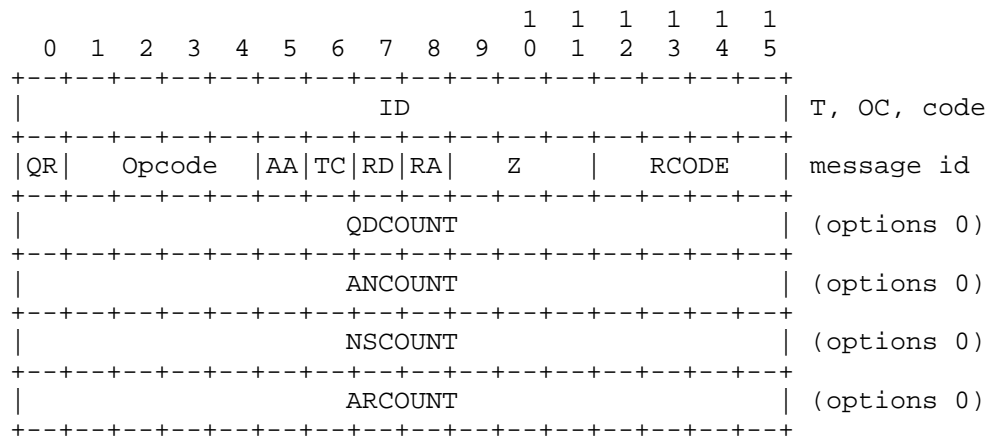


Figure 11: DNS Header vs. CoAP Message

In general, for any pair of protocols, one of the protocols can very well have been designed in a way that enables an attacker to cause the generation of replies that look like messages of the other protocol. It is often much harder to ensure or prove the absence of viable attacks than to generate examples that may not yet completely enable an attack but might be further developed by more creative minds. Cross-protocol attacks can therefore only be completely

mitigated if endpoints don't authorize actions desired by an attacker just based on trusting the source IP address of a packet. Conversely, a NoSec environment that completely relies on a firewall for CoAP security not only needs to firewall off the CoAP endpoints but also all other endpoints that might be incited to send UDP messages to CoAP endpoints using some other UDP-based protocol.

In addition to the considerations above, the security considerations for DTLS with respect to cross-protocol attacks apply. E.g., if the same DTLS security association ("connection") is used to carry data of multiple protocols, DTLS no longer provides protection against cross-protocol attacks between these protocols.

## 12. IANA Considerations

### 12.1. CoAP Code Registry

This document defines a registry for the values of the Code field in the CoAP header. The name of the registry is "CoAP Codes".

All values are assigned by sub-registries according to the following ranges:

|         |   |
|---------|---|
| 0       | Indicates an empty message (see Section 4.1).   |
| 1-31    | Indicates a request. Values in this range are assigned by the "CoAP Method Codes" sub-registry (see Section 12.1.1).    |
| 32-63   | Reserved  |
| 64-191  | Indicates a response. Values in this range are assigned by the "CoAP Response Codes" sub-registry (see Section 12.1.2). |
| 192-255 | Reserved  |

#### 12.1.1. Method Codes

The name of the sub-registry is "CoAP Method Codes".

Each entry in the sub-registry must include the Method Code in the range 1-31, the name of the method, and a reference to the method's documentation.

Initial entries in this sub-registry are as follows:

| Code | Name   | Reference |
|------|--------|-----------|
| 1    | GET    | [RFCXXXX] |
| 2    | POST   | [RFCXXXX] |
| 3    | PUT    | [RFCXXXX] |
| 4    | DELETE | [RFCXXXX] |

Table 2: CoAP Method Codes

All other Method Codes are Unassigned.

The IANA policy for future additions to this registry is "IETF Review" as described in [RFC5226].

The documentation of a method code should specify the semantics of a request with that code, including the following properties:

- o The response codes the method returns in the success case.
- o Whether the method is idempotent, safe, or both.

#### 12.1.2. Response Codes

The name of the sub-registry is "CoAP Response Codes".

Each entry in the sub-registry must include the Response Code in the range 64-191, a description of the Response Code, and a reference to the Response Code's documentation.

Initial entries in this sub-registry are as follows:

| Code | Description                   | Reference |
|------|-------------------------------|-----------|
| 65   | 2.01 Created                  | [RFCXXXX] |
| 66   | 2.02 Deleted                  | [RFCXXXX] |
| 67   | 2.03 Valid                    | [RFCXXXX] |
| 68   | 2.04 Changed                  | [RFCXXXX] |
| 69   | 2.05 Content                  | [RFCXXXX] |
| 128  | 4.00 Bad Request              | [RFCXXXX] |
| 129  | 4.01 Unauthorized             | [RFCXXXX] |
| 130  | 4.02 Bad Option               | [RFCXXXX] |
| 131  | 4.03 Forbidden                | [RFCXXXX] |
| 132  | 4.04 Not Found                | [RFCXXXX] |
| 133  | 4.05 Method Not Allowed       | [RFCXXXX] |
| 134  | 4.06 Not Acceptable           | [RFCXXXX] |
| 140  | 4.12 Precondition Failed      | [RFCXXXX] |
| 141  | 4.13 Request Entity Too Large | [RFCXXXX] |
| 143  | 4.15 Unsupported Media Type   | [RFCXXXX] |
| 160  | 5.00 Internal Server Error    | [RFCXXXX] |
| 161  | 5.01 Not Implemented          | [RFCXXXX] |
| 162  | 5.02 Bad Gateway              | [RFCXXXX] |
| 163  | 5.03 Service Unavailable      | [RFCXXXX] |
| 164  | 5.04 Gateway Timeout          | [RFCXXXX] |
| 165  | 5.05 Proxying Not Supported   | [RFCXXXX] |

Table 3: CoAP Response Codes

The Response Codes 96-127 are Reserved for future use. All other Response Codes are Unassigned.

The IANA policy for future additions to this registry is "IETF Review" as described in [RFC5226].

The documentation of a response code should specify the semantics of a response with that code, including the following properties:

- o The methods the response code applies to.
- o Whether payload is required, optional or not allowed.
- o The semantics of the payload. For example, the payload of a 2.05 (Content) response is a representation of the target resource; the payload in an error response is a human-readable diagnostic message.
- o The format of the payload. For example, the format in a 2.05 (Content) response is indicated by the Content-Type Option; the

format of the payload in an error response is always Net-Unicode text.

- o Whether the response is cacheable according to the freshness model.
- o Whether the response is validatable according to the validation model.
- o Whether the response causes a cache to mark responses stored for the request URI as not fresh.

#### 12.2. Option Number Registry

This document defines a registry for the Option Numbers used in CoAP options. The name of the registry is "CoAP Option Numbers".

Each entry in the registry must include the Option Number, the name of the option and a reference to the option's documentation.

Initial entries in this registry are as follows:



| Number | Name           | Reference |
|--------|----------------|-----------|
| 0      | (Reserved)     |           |
| 1      | Content-Type   | [RFCXXXX] |
| 2      | Max-Age        | [RFCXXXX] |
| 3      | Proxy-Uri      | [RFCXXXX] |
| 4      | ETag           | [RFCXXXX] |
| 5      | Uri-Host       | [RFCXXXX] |
| 6      | Location-Path  | [RFCXXXX] |
| 7      | Uri-Port       | [RFCXXXX] |
| 8      | Location-Query | [RFCXXXX] |
| 9      | Uri-Path       | [RFCXXXX] |
| 10     | (Unassigned)   |           |
| 11     | Token          | [RFCXXXX] |
| 12     | Accept         | [RFCXXXX] |
| 13     | If-Match       | [RFCXXXX] |
| 14     | (Unassigned)   |           |
| 15     | Uri-Query      | [RFCXXXX] |
| 16-20  | (Unassigned)   |           |
| 21     | If-None-Match  | [RFCXXXX] |
| 22-43  | (Unassigned)   |           |
| 44     | (Reserved)     |           |
| 45     | (Unassigned)   |           |
| 46     | (Reserved)     |           |
| 47     | (Unassigned)   |           |
| 48     | (Reserved)     |           |
| 49-    | (Unassigned)   |           |

Table 4: CoAP Option Numbers

The IANA policy for future additions to this registry is "IETF Review" as described in [RFC5226].

The documentation of an Option Number should specify the semantics of an option with that number, including the following properties:

- o The meaning of the option in a request.
- o The meaning of the option in a response.
- o Whether the option is critical or elective, as determined by the Option Number.
- o The format and length of the option's value.

- o Whether the option must occur at most once or whether it can occur multiple times.
- o The default value, if any. For a critical option with a default value, a discussion on how the default value enables processing by implementations not implementing the critical option (Section 5.4.3). For options with numbers that are a multiple of 14, the default value MUST be empty.

### 12.3. Media Type Registry

Media types are identified by a string, such as "application/xml" [RFC2046]. In order to minimize the overhead of using these media types to indicate the format of payloads, this document defines a registry for a subset of Internet media types to be used in CoAP and assigns each a numeric identifier. The name of the registry is "CoAP Media Types".

Each entry in the registry must include the media type registered with IANA, the numeric identifier in the range 0-65535 to be used for that media type in CoAP, the content-encoding associated with this identifier, and a reference to a document describing what a payload with that media type means semantically.

CoAP does not include a way to convey content-encoding information with a request or response, and for that reason the content-encoding is also specified for each identifier (if any). If multiple content-encodings will be used with a media type, then a separate identifier for each is to be registered.

Initial entries in this registry are as follows:

| Media type                   | Encoding | Id. | Reference                   |
|------------------------------|----------|-----|-----------------------------|
| text/plain;<br>charset=utf-8 | -        | 0   | [RFC2046][RFC3676][RFC5147] |
| application/<br>link-format  | -        | 40  | [I-D.ietf-core-link-format] |
| application/xml              | -        | 41  | [RFC3023]                   |
| application/<br>octet-stream | -        | 42  | [RFC2045][RFC2046]          |
| application/exi              | -        | 47  | [EXIMIME]                   |
| application/json             | -        | 50  | [RFC4627]                   |

Table 5: CoAP Media Types

The identifiers between 201 and 255 inclusive are reserved for Private Use. All other identifiers are Unassigned.

Because the name space of single-byte identifiers is so small, the IANA policy for future additions in the range 0-200 inclusive to the registry is "Expert Review" as described in [RFC5226]. The IANA policy for additions in the range 256-65535 inclusive is "First Come First Served" as described in [RFC5226].

In machine to machine applications, it is not expected that generic Internet media types such as text/plain, application/xml or application/octet-stream are useful for real applications in the long term. It is recommended that M2M applications making use of CoAP will request new Internet media types from IANA indicating semantic information about how to create or parse a payload. For example, a Smart Energy application payload carried as XML might request a more specific type like application/se+xml or application/se+exi.

#### 12.4. URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap". The registration request complies with [RFC4395].

URI scheme name.  
coap

Status.  
Permanent.

URI scheme syntax.  
Defined in Section 6.1 of [RFCXXXX].

URI scheme semantics.  
The "coap" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP). The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "http" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.  
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.

The scheme is used by CoAP endpoints to access CoAP resources.

Interoperability considerations.

None.

Security considerations.

See Section 11.1 of [RFCXXXX].

Contact.

IETF Chair <chair@ietf.org>

Author/Change controller.

IESG <iesg@ietf.org>

References.

[RFCXXXX]

## 12.5. Secure URI Scheme Registration

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coaps". The registration request complies with [RFC4395].

URI scheme name.

coaps

Status.

Permanent.

URI scheme syntax.

Defined in Section 6.2 of [RFCXXXX].

URI scheme semantics.

The "coaps" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using Datagram Transport Layer Security (DTLS) for transport security. The resources can be located by contacting the governing CoAP server and operated on by sending CoAP requests to the server. This scheme can thus be compared to the "https" URI scheme [RFC2616]. See Section 6 of [RFCXXXX] for the details of operation.

Encoding considerations.

The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e. internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.  
The scheme is used by CoAP endpoints to access CoAP resources using DTLS.

Interoperability considerations.  
None.

Security considerations.  
See Section 11.1 of [RFCXXXX].

Contact.  
IETF Chair <chair@ietf.org>

Author/Change controller.  
IESG <iesg@ietf.org>

References.  
[RFCXXXX]

#### 12.6. Service Name and Port Number Registration

One of the functions of CoAP is resource discovery: a CoAP client can ask a CoAP server about the resources offered by it (see Section 7). To enable resource discovery just based on the knowledge of an IP address, the CoAP port for resource discovery needs to be standardized.

IANA has assigned the port number 5683 and the service name "coap", in accordance with [RFC6335].

Besides unicast, CoAP can be used with both multicast and anycast.

Service Name.  
coap

Transport Protocol.  
UDP

Assignee.  
IESG <iesg@ietf.org>

Contact.  
IETF Chair <chair@ietf.org>

Description.  
Constrained Application Protocol (CoAP)

Reference.  
[RFCXXXX]

Port Number.  
5683

#### 12.7. Secure Service Name and Port Number Registration

CoAP resource discovery may also be provided using the DTLS-secured CoAP "coaps" scheme. Thus the CoAP port for secure resource discovery needs to be standardized.

This document requests the assignment of the port number [IANA\_TBD\_PORT] and the service name "coaps", in accordance with [RFC6335].

Besides unicast, DTLS-secured CoAP can be used with anycast.

Service Name.  
coaps

Transport Protocol.  
UDP

Assignee.  
IESG <iesg@ietf.org>

Contact.  
IETF Chair <chair@ietf.org>

Description.  
DTLS-secured CoAP

Reference.  
[RFCXXXX]

Port Number.  
[IANA\_TBD\_PORT]

#### 12.8. Multicast Address Registration

Section 8, "Multicast CoAP", defines the use of multicast. This document requests the assignment of the following multicast addresses for use by CoAP nodes:

IPv4 -- "All CoAP Nodes" address [TBD1], from the IPv4 Multicast Address Space Registry. As the address is used for discovery that may span beyond a single network, it should come from the Internetwork Control Block (224.0.1.x, RFC 5771).

IPv6 -- "All CoAP Nodes" address [TBD2], from the IPv6 Multicast Address Space Registry, in the Variable Scope Multicast Addresses space (RFC3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to.

[The explanatory text to be removed upon allocation of the addresses, except for the note about the distinct multicast addresses.]

### 13. Acknowledgements

Special thanks to Peter Bigot, Esko Dijk and Cullen Jennings for substantial contributions to the ideas and text in the document, along with countless detailed reviews and discussions.

Thanks to Ed Beroaset, Angelo P. Castellani, Gilbert Clark, Robert Cragie, Esko Dijk, Lisa Dussealt, Thomas Fossati, Tom Herbst, Richard Kelsey, Ari Keranen, Matthias Kovatsch, Salvatore Loreto, Kerry Lynn, Alexey Melnikov, Guido Moritz, Petri Mutka, Colin O'Flynn, Charles Palmer, Adriano Pezzuto, Robert Quattlebaum, Akbar Rahman, Eric Rescorla, David Ryan, Szymon Sasin, Michael Scharf, Dale Seed, Robby Simpson, Peter van der Stok, Michael Stuber, Linyi Tian, Gilman Tolle, Matthieu Vial and Alper Yegin for helpful comments and discussions that have shaped the document.

Some of the text has been lifted from the working documents of the IETF httpbis working group.

### 14. References

#### 14.1. Normative References

- [I-D.farrell-decade-ni]  
Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", draft-farrell-decade-ni-09 (work in progress), July 2012.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-14 (work in progress), June 2012.

- [I-D.ietf-tls-oob-pubkey]  
Wouters, P., Gilmore, J., Weiler, S., Kivinen, T., and H. Tschofenig, "Out-of-Band Public Key Validation for Transport Layer Security", draft-ietf-tls-oob-pubkey-04 (work in progress), July 2012.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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, June 1999.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [RFC3602] Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec", RFC 3602, September 2003.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.
- [RFC4309] Housley, R., "Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)", RFC 4309, December 2005.



- [RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.
- [RFC4835] Manral, V., "Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 4835, April 2007.
- [RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers for the text/plain Media Type", RFC 5147, April 2008.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

## 14.2. Informative References

- [EUI64]    "GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64) REGISTRATION AUTHORITY", April 2010, <<http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>>.
- [EXIMIME]    "Efficient XML Interchange (EXI) Format 1.0", December 2009, <<http://www.w3.org/TR/2009/CR-exi-20091208/#mediaTypeRegistration>>.
- [I-D.allman-tcpm-rto-consider]  
Allman, M., "Retransmission Timeout Considerations", draft-allman-tcpm-rto-consider-01 (work in progress), May 2012.
- [I-D.eggert-core-congestion-control]  
Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-08 (work in progress), February 2012.
- [I-D.ietf-httpbis-p1-messaging]  
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part 1: Message Routing and Syntax", draft-ietf-httpbis-p1-messaging-20 (work in progress), July 2012.
- [I-D.kivinen-ipsecme-ikev2-minimal]  
Kivinen, T., "Minimal IKEv2", draft-kivinen-ipsecme-ikev2-minimal-00 (work in progress), February 2011.
- [I-D.mcgrew-tls-aes-ccm]  
McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for TLS", draft-mcgrew-tls-aes-ccm-03 (work in progress), February 2012.
- [I-D.mcgrew-tls-aes-ccm-ecc]  
McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM ECC Cipher Suites for TLS", draft-mcgrew-tls-aes-ccm-ecc-02 (work in progress), October 2011.

- [REST]     Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <[http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)>.
- [RFC0793]   Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC2818]   Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC3264]   Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC3542]   Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- [RFC4492]   Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.
- [RFC4627]   Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC4944]   Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5405]   Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [RFC6120]   Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6335]   Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

## Appendix A. Examples

This section gives a number of short examples with message flows for GET requests. These examples demonstrate the basic operation, the

operation in the presence of retransmissions, and multicast.

Figure 12 shows a basic GET request causing a piggy-backed response: The client sends a Confirmable GET request for the resource `coap://server/temperature` to the server with a Message ID of `0x7d34`. The request includes one Uri-Path Option (Delta 0 + 9 = 9, Length 11, Value "temperature"); the Token is left at its default value (empty). This request is a total of 16 bytes long. A 2.05 (Content) response is returned in the Acknowledgement message that acknowledges the Confirmable request, echoing both the Message ID `0x7d34` and the (implicitly empty) Token value. The response includes a Payload of "22.3 C" and is 10 bytes long.

Client    Server

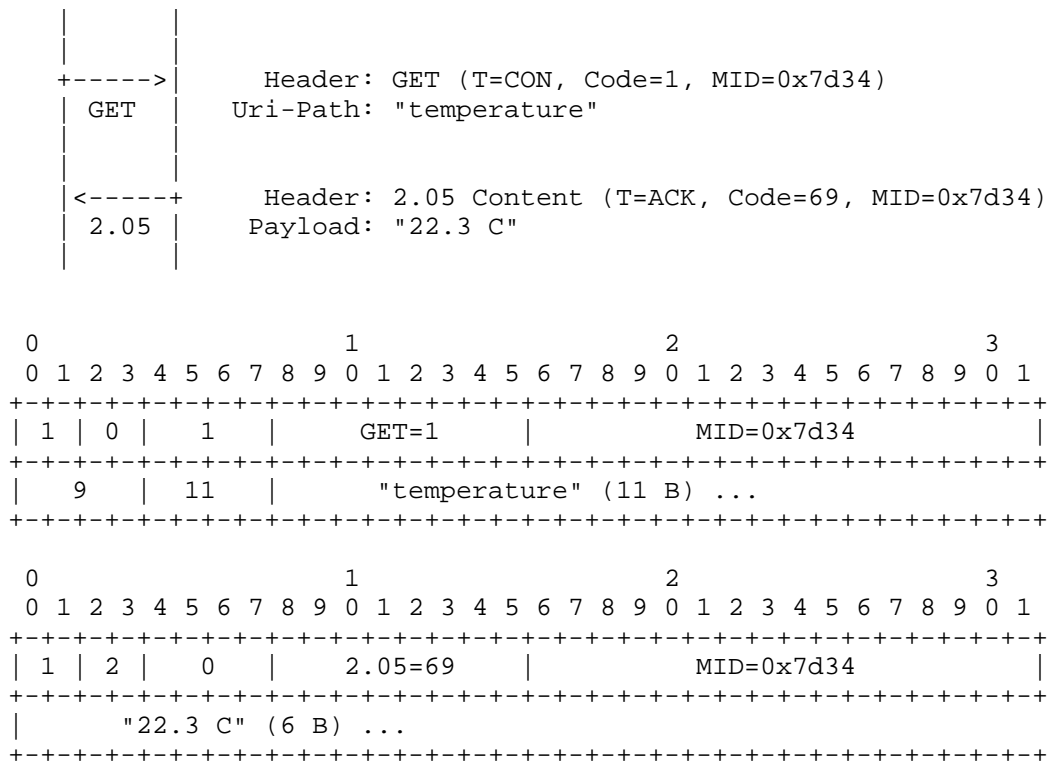


Figure 12: Confirmable request; piggy-backed response

Figure 13 shows a similar example, but with the inclusion of an explicit Token Option (Delta 9 + 2 = 11, Length 1, Value `0x20`) in the request and (Delta 11 + 0 = 11) in the response, increasing the sizes to 18 and 12 bytes, respectively.

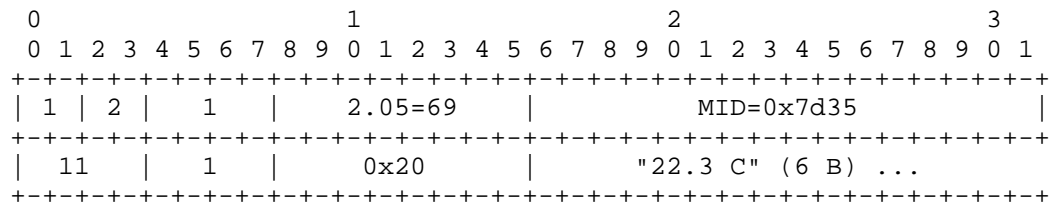
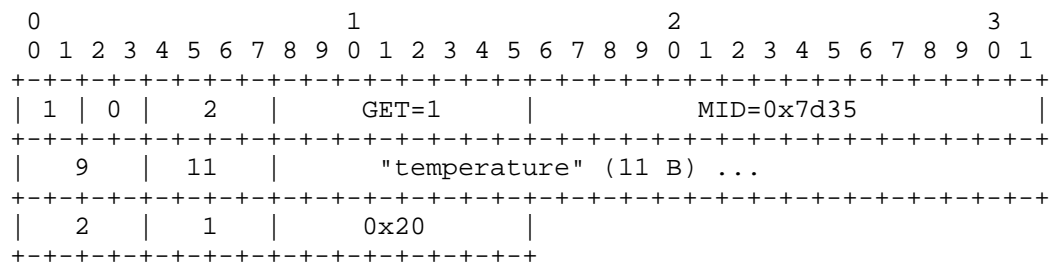
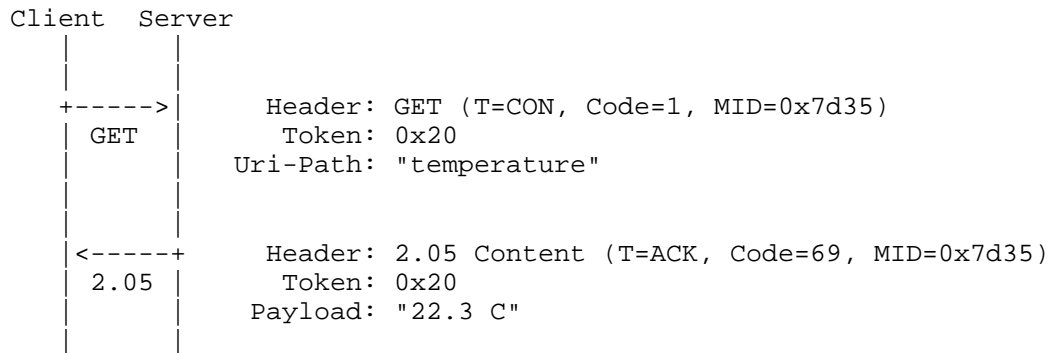


Figure 13: Confirmable request; piggy-backed response

In Figure 14, the Confirmable GET request is lost. After ACK\_TIMEOUT seconds, the client retransmits the request, resulting in a piggy-backed response as in the previous example.

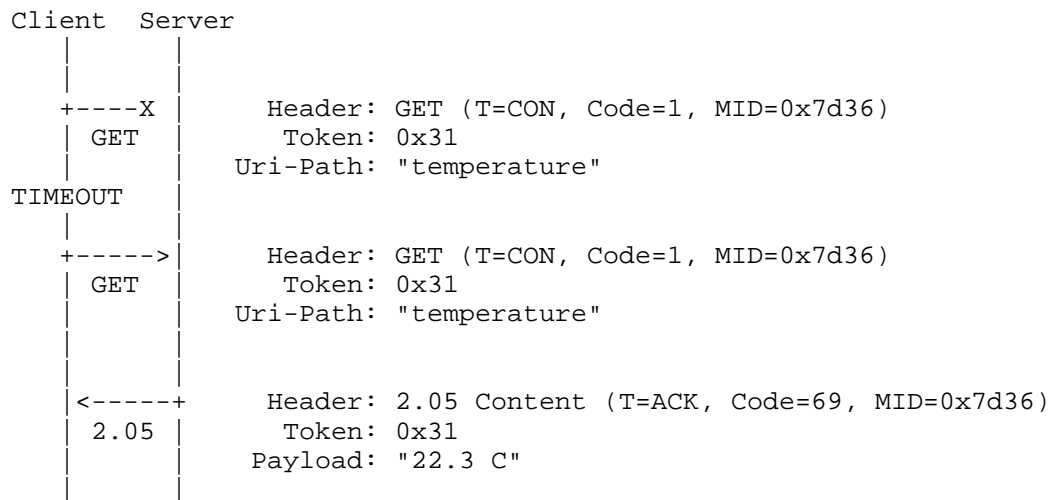


Figure 14: Confirmable request (retransmitted); piggy-backed response

In Figure 15, the first Acknowledgement message from the server to the client is lost. After ACK\_TIMEOUT seconds, the client retransmits the request.

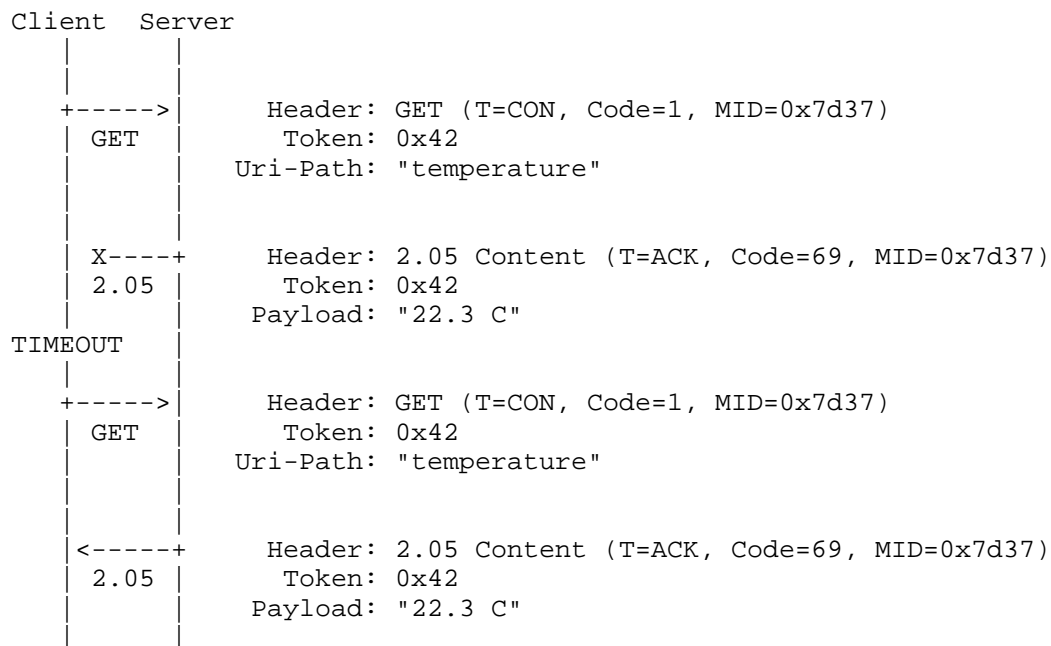


Figure 15: Confirmable request; piggy-backed response (retransmitted)

In Figure 16, the server acknowledges the Confirmable request and sends a 2.05 (Content) response separately in a Confirmable message. Note that the Acknowledgement message and the Confirmable response do not necessarily arrive in the same order as they were sent. The client acknowledges the Confirmable response.

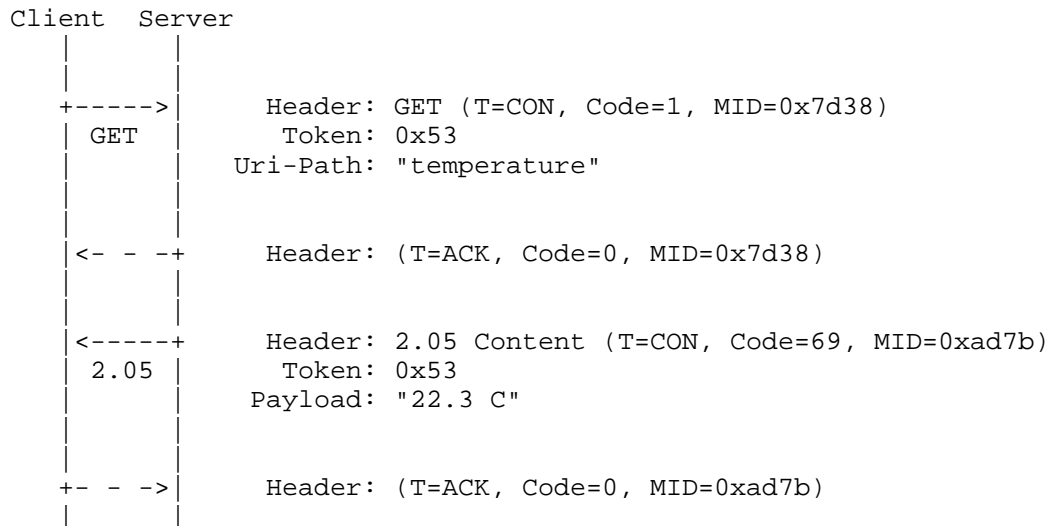


Figure 16: Confirmable request; separate response

Figure 17 shows an example where the client loses its state (e.g., crashes and is rebooted) right after sending a Confirmable request, so the separate response arriving some time later comes unexpected. In this case, the client rejects the Confirmable response with a Reset message. Note that the unexpected ACK is silently ignored.

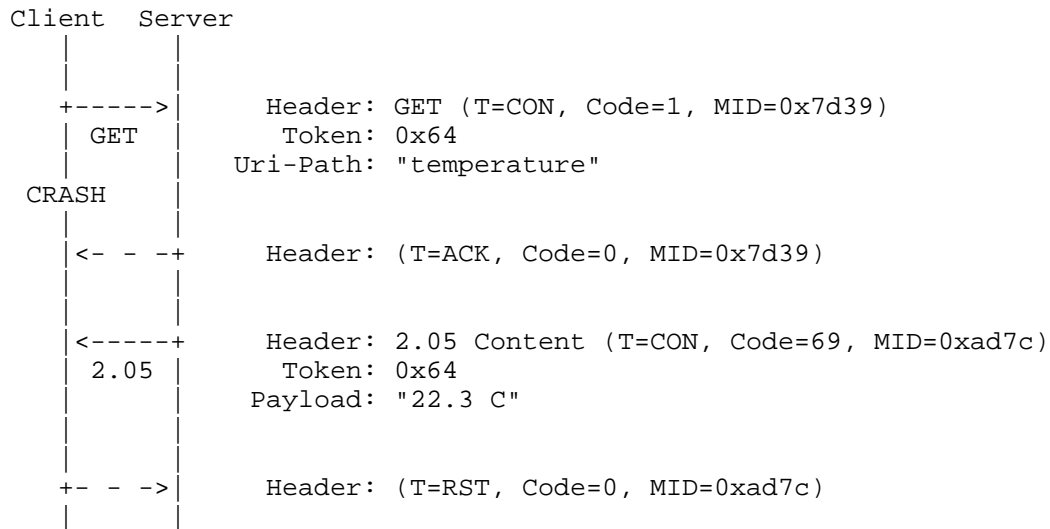


Figure 17: Confirmable request; separate response (unexpected)

Figure 18 shows a basic GET request where the request and the response are non-confirmable, so both may be lost without notice.

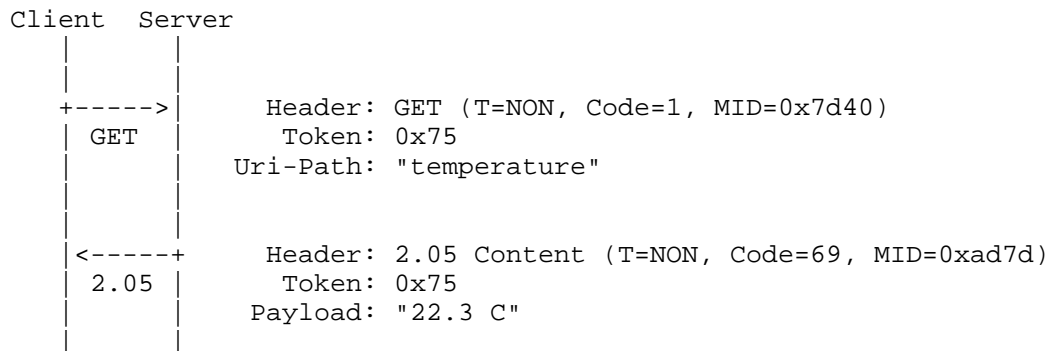


Figure 18: Non-confirmable request; Non-confirmable response

In Figure 19, the client sends a Non-confirmable GET request to a multicast address: all nodes in link-local scope. There are 3 servers on the link: A, B and C. Servers A and B have a matching resource, therefore they send back a Non-confirmable 2.05 (Content) response. The response sent by B is lost. C does not have matching response, therefore it sends a Non-confirmable 4.04 (Not Found) response.



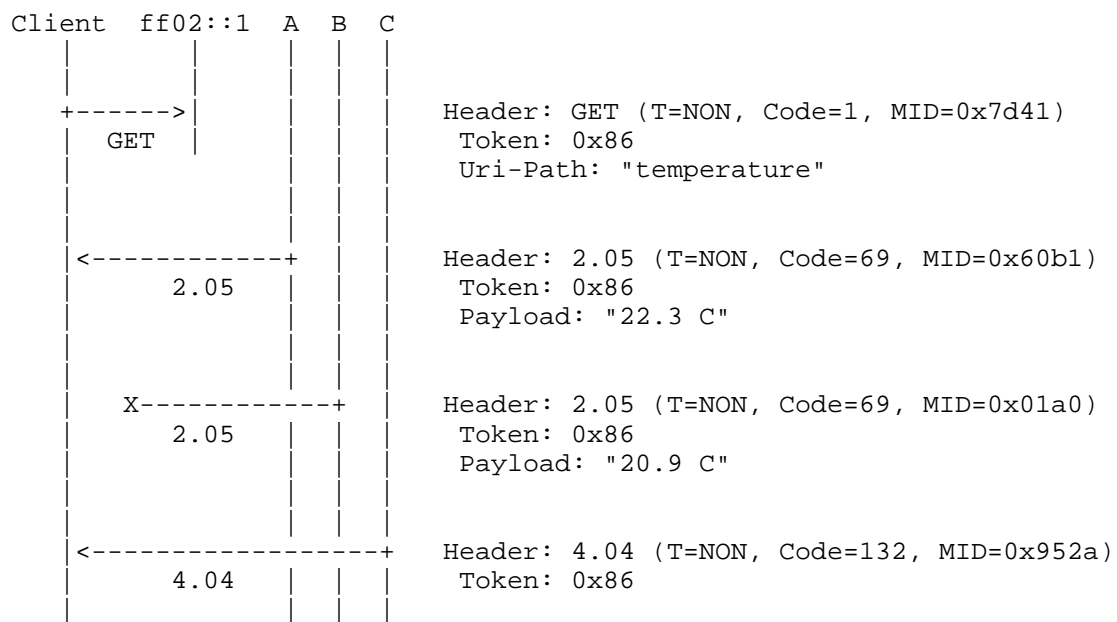


Figure 19: Non-confirmable request (multicast); Non-confirmable response

## Appendix B. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them.

- o `coap://[2001:db8::2:1]/`  
     Destination IP Address = [2001:db8::2:1]  
     Destination UDP Port = 5683
- o `coap://example.net/`  
     Destination IP Address = [2001:db8::2:1]  
     Destination UDP Port = 5683  
     Uri-Host = "example.net"
- o `coap://example.net/.well-known/core`

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = 5683

Uri-Host = "example.net"

Uri-Path = ".well-known"

Uri-Path = "core"

- o coap://  
xn--18j4d.example/%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF

Destination IP Address = [2001:db8::2:1]

Destination UDP Port = 5683

Uri-Host = "xn--18j4d.example"

Uri-Path = the string composed of the Unicode characters U+3053 U+3093 U+306b U+3061 U+306f, usually represented in UTF-8 as E38193E38293E381ABE381A1E381AF hexadecimal

- o coap://198.51.100.1:61616//%2F//?%2F%2F&?%26

Destination IP Address = 198.51.100.1

Destination UDP Port = 61616

Uri-Path = ""

Uri-Path = "/"

Uri-Path = ""

Uri-Path = ""

Uri-Query = "//"

Uri-Query = "?&"

## Appendix C. Changelog

Changed from ietf-10 to ietf-11:

- o Expanded section 4.8 on Transmission Parameters, and used the derived values defined there (#201). Changed parameter names to

be shorter and more to the point.

- o Several more small editorial changes, clarifications and improvements have been made.

Changed from ietf-09 to ietf-10:

- o Option deltas are restricted to 0 to 14; the option delta 15 is used exclusively for the end-of-options marker (#239).
- o Option numbers that are a multiple of 14 are not reserved, but are required to have an empty default value (#212).
- o Fixed misleading language that was introduced in 5.10.2 in coap-07 re Uri-Host and Uri-Port (#208).
- o Segments and arguments can have a length of zero characters (#213).
- o The Location-\* options describe together describe one location. The location is a relative URI, not an "absolute path URI" (#218).
- o The value of the Location-Path Option must not be '.' or '..' (#218).
- o Added a sentence on constructing URIs from Location-\* options (#231).
- o Reserved option numbers for future Location-\* options (#230).
- o Fixed response codes with payload inconsistency (#233).
- o Added advice on default values for critical options (#207).
- o Clarified use of identifiers in RawPublicKey Mode Provisioning (#222).
- o Moved "Securing CoAP" out of the "Security Considerations" (#229).
- o Added "All CoAP Nodes" multicast addresses to "IANA Considerations" (#216).
- o Over 100 small editorial changes, clarifications and improvements have been made.

Changed from ietf-08 to ietf-09:

- o Improved consistency of statements about RST on NON: RST is a valid response to a NON message (#183).
- o Clarified that the protocol constants can be configured for specific application environments.
- o Added implementation note recommending piggy-backing whenever possible (#182).
- o Added a content-encoding column to the media type registry (#181).
- o Minor improvements to Appendix D.
- o Added text about multicast response suppression (#177).
- o Included the new End-of-options Marker (#176).
- o Added a reference to draft-ietf-tls-oob-pubkey and updated the RPK text accordingly.

Changed from ietf-07 to ietf-08:

- o Clarified matching rules for messages (#175)
- o Fixed a bug in Section 8.2.2 on Etags (#168)
- o Added an IP address spoofing threat analysis contribution (#167)
- o Re-focused the security section on raw public keys (#166)
- o Added an 4.06 error to Accept (#165)

Changed from ietf-06 to ietf-07:

- o application/link-format added to Media types registration (#160)
- o Moved content-type attribute to the document from link-format.
- o Added coaps scheme and DTLS-secured CoAP default port (#154)
- o Allowed 0-length Content-type options (#150)
- o Added congestion control recommendations (#153)
- o Improved text on PUT/POST response payloads (#149)
- o Added an Accept option for content-negotiation (#163)

- o Added If-Match and If-None-Match options (#155)
- o Improved Token Option explanation (#147)
- o Clarified mandatory to implement security (#156)
- o Added first come first server policy for 2-byte Media type codes (#161)
- o Clarify matching rules for messages and tokens (#151)
- o Changed OPTIONS and TRACE to always return 501 in HTTP-CoAP mapping (#164)

Changed from ietf-05 to ietf-06:

- o HTTP mapping section improved with the minimal protocol standard text for CoAP-HTTP and HTTP-CoAP forward proxying (#137).
- o Eradicated percent-encoding by including one Uri-Query Option per &-delimited argument in a query.
- o Allowed RST message in reply to a NON message with unexpected token (#135).
- o Cache Invalidation only happens upon successful responses (#134).
- o 50% jitter added to the initial retransmit timer (#142).
- o DTLS cipher suites aligned with ZigBee IP, DTLS clarified as default CoAP security mechanism (#138, #139)
- o Added a minimal reference to draft-kivinen-ipsecme-ikev2-minimal (#140).
- o Clarified the comparison of UTF-8s (#136).
- o Minimized the initial media type registry (#101).

Changed from ietf-04 to ietf-05:

- o Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- o GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).

- o Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- o Improved message deduplication rules (#106).
- o Section added on message size implementation considerations (#103).
- o Clarification made on human readable error payloads (#109).
- o Definition of CoAP methods improved (#108).
- o Max-Age removed from requests (#107).
- o Clarified uniqueness of tokens (#112).
- o Location-Query Option added (#113).
- o ETag length set to 1-8 bytes (#123).
- o Clarified relation between elective/critical and option numbers (#110).
- o Defined when to update Version header field (#111).
- o URI scheme registration improved (#102).
- o Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- o Major document reorganization (#51, #63, #71, #81).
- o Max-age length set to 0-4 bytes (#30).
- o Added variable unsigned integer definition (#31).
- o Clarification made on human readable error payloads (#50).
- o Definition of POST improved (#52).
- o Token length changed to 0-8 bytes (#53).
- o Section added on multiplexing CoAP, DTLS and STUN (#56).
- o Added cross-protocol attack considerations (#61).

- o Used new Immediate/Deferred response definitions (#73).
- o Improved request/response matching rules (#74).
- o Removed unnecessary media types and added recommendations for their use in M2M (#76).
- o Response codes changed to base 32 coding, new Y.XX naming (#77).
- o References updated as per AD review (#79).
- o IANA section completed (#80).
- o Proxy-Uri Option added to disambiguate between proxy and non-proxy requests (#82).
- o Added text on critical options in cached states (#83).
- o HTTP mapping sections improved (#88).
- o Added text on reverse proxies (#72).
- o Some security text on multicast added (#54).
- o Trust model text added to introduction (#58, #60).
- o AES-CCM vs. AES-CCB text added (#55).
- o Text added about device capabilities (#59).
- o DTLS section improvements (#87).
- o Caching semantics aligned with RFC2616 (#78).
- o Uri-Path Option split into multiple path segments.
- o MAX\_RETRANSMIT changed to 4 to adjust for RESPONSE\_TIME = 2.

Changes from ietf-02 to ietf-03:

- o Token Option and related use in asynchronous requests added (#25).
- o CoAP specific error codes added (#26).
- o Erroring out on unknown critical options changed to a MUST (#27).
- o Uri-Query Option added.

- o Terminology and definitions of URIs improved.
- o Security section completed (#22).

Changes from ietf-01 to ietf-02:

- o Sending an error on a critical option clarified (#18).
- o Clarification on behavior of PUT and idempotent operations (#19).
- o Use of Uri-Authority clarified along with server processing rules; Uri-Scheme Option removed (#20, #23).
- o Resource discovery section removed to a separate CoRE Link Format draft (#21).
- o Initial security section outline added.

Changes from ietf-00 to ietf-01:

- o New cleaner transaction message model and header (#5).
- o Removed subscription while being designed (#1).
- o Section 2 re-written (#3).
- o Text added about use of short URIs (#4).
- o Improved header option scheme (#5, #14).
- o Date option removed while being designed (#6).
- o New text for CoAP default port (#7).
- o Completed proxying section (#8).
- o Completed resource discovery section (#9).
- o Completed HTTP mapping section (#10).
- o Several new examples added (#11).
- o URI split into 3 options (#12).
- o MIME type defined for link-format (#13, #16).
- o New text on maximum message size (#15).



- o Location Option added.

Changes from shelby-01 to ietf-00:

- o Removed the TCP binding section, left open for the future.
- o Fixed a bug in the example.
- o Marked current Sub/Notify as (Experimental) while under WG discussion.
- o Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- o Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- o Removed the Uri-code Option as different URI encoding schemes are being discussed.
- o Changed the rel= field to desc= for resource discovery.
- o Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.
- o Made the URI slash optimization and method idempotence MUSTs
- o Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- o Unified the message header and added a notify message type.
- o Renamed methods with HTTP names and removed the NOTIFY method.
- o Added a number of options field to the header.
- o Combines the Option Type and Length into an 8-bit field.
- o Added the magic byte header.
- o Added new ETag Option.
- o Added new Date Option.
- o Added new Subscription Option.

- o Completed the HTTP Code - CoAP Code mapping table appendix.
- o Completed the Content-type Identifier appendix and tables.
- o Added more simplifications for URI support.
- o Initial subscription and discovery sections.
- o A Flag requirements simplified.

Authors' Addresses

Zach Shelby  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
Finland

Phone: +358407796297  
Email: zach@sensinode.com

Klaus Hartke  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63905  
Fax: +49-421-218-7000  
Email: hartke@tzi.org

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Fax: +49-421-218-7000  
Email: cabo@tzi.org

Brian Frank  
SkyFoundry  
Richmond, VA  
USA

Phone:  
Email: [brian@skyfoundry.com](mailto:brian@skyfoundry.com)



CoRE Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: March 16, 2015

A. Rahman, Ed.  
InterDigital Communications, LLC  
E. Dijk, Ed.  
Philips Research  
September 12, 2014

Group Communication for CoAP  
draft-ietf-core-groupcomm-25

Abstract

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for constrained devices and constrained networks. It is anticipated that constrained devices will often naturally operate in groups (e.g., in a building automation scenario all lights in a given room may need to be switched on/off as a group). This specification defines how the CoAP protocol should be used in a group communication context. An approach for using CoAP on top of IP multicast is detailed based on both existing CoAP functionality as well as new features introduced in this specification. Also, various use cases and corresponding protocol flows are provided to illustrate important concepts. Finally, guidance is provided for deployment in various network topologies.

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 March 16, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .  | 3  |
| 1.1. Background . . . . .  | 3  |
| 1.2. Scope . . . . .   | 3  |
| 1.3. Conventions and Terminology . . . . .                           | 4  |
| 2. Protocol Considerations . . . . .                                 | 5  |
| 2.1. IP Multicast Background . . . . .                               | 5  |
| 2.2. Group Definition and Naming . . . . .                           | 6  |
| 2.3. Port and URI Configuration . . . . .                            | 7  |
| 2.4. RESTful Methods . . . . .                                       | 9  |
| 2.5. Request and Response Model . . . . .                            | 9  |
| 2.6. Membership Configuration . . . . .                              | 10 |
| 2.6.1. Background . . . . .  | 10 |
| 2.6.2. Membership Configuration RESTful Interface . . . . .          | 11 |
| 2.7. Request Acceptance and Response Suppression Rules . . . . .     | 17 |
| 2.8. Congestion Control . . . . .                                    | 19 |
| 2.9. Proxy Operation . . . . .                                       | 20 |
| 2.10. Exceptions . . . . .   | 21 |
| 3. Use Cases and Corresponding Protocol Flows . . . . .              | 22 |
| 3.1. Introduction . . . . .  | 22 |
| 3.2. Network Configuration . . . . .                                 | 22 |
| 3.3. Discovery of Resource Directory . . . . .                       | 24 |
| 3.4. Lighting Control . . . . .                                      | 26 |
| 3.5. Lighting Control in MLD Enabled Network . . . . .               | 30 |
| 3.6. Commissioning the Network Based On Resource Directory . . . . . | 31 |
| 4. Deployment Guidelines . . . . .                                   | 32 |
| 4.1. Target Network Topologies . . . . .                             | 32 |
| 4.2. Networks Using the MLD Protocol . . . . .                       | 33 |
| 4.3. Networks Using RPL Multicast Without MLD . . . . .              | 33 |
| 4.4. Networks Using MPL Forwarding Without MLD . . . . .             | 34 |
| 4.5. 6LoWPAN Specific Guidelines for the 6LBR . . . . .              | 35 |
| 5. Security Considerations . . . . .                                 | 35 |
| 5.1. Security Configuration . . . . .                                | 35 |
| 5.2. Threats . . . . .   | 36 |
| 5.3. Threat Mitigation . . . . .                                     | 36 |
| 5.3.1. WiFi Scenario . . . . .                                       | 37 |
| 5.3.2. 6LoWPAN Scenario . . . . .                                    | 37 |

|  |    |
|--|----|
| 5.3.3. Future Evolution . . . . .                        | 37 |
| 5.4. Monitoring Considerations . . . . .                 | 38 |
| 5.4.1. General Monitoring . . . . .                      | 38 |
| 5.4.2. Pervasive Monitoring . . . . .                    | 38 |
| 6. IANA Considerations . . . . .                         | 39 |
| 6.1. New 'core.gp' Resource Type . . . . .               | 39 |
| 6.2. New 'coap-group+json' Internet Media Type . . . . . | 39 |
| 7. Acknowledgements . . . . .                            | 41 |
| 8. References . . . . .                                  | 41 |
| 8.1. Normative References . . . . .                      | 41 |
| 8.2. Informative References . . . . .                    | 43 |
| Appendix A. Multicast Listener Discovery (MLD) . . . . . | 44 |
| Appendix B. Change Log . . . . .                         | 44 |
| Authors' Addresses . . . . .                             | 57 |

## 1. Introduction

### 1.1. Background

Constrained Application Protocol (CoAP) is a Representational State Transfer (REST) based web transfer protocol for resource constrained devices operating in an IP network [RFC7252]. CoAP has many similarities to HTTP [RFC7230] but also has some key differences. Constrained devices can be large in numbers, but are often related to each other in function or by location. For example, all the light switches in a building may belong to one group and all the thermostats may belong to another group. Groups may be pre-configured before deployment or dynamically formed during operation. If information needs to be sent to or received from a group of devices, group communication mechanisms can improve efficiency and latency of communication and reduce bandwidth requirements for a given application. HTTP does not support any equivalent functionality to CoAP group communication.

### 1.2. Scope

Group communication involves a one-to-many relationship between CoAP endpoints. Specifically, a single CoAP client can simultaneously get (or set) resources from multiple CoAP servers using CoAP over IP multicast. An example would be a CoAP light switch turning on/off multiple lights in a room with a single CoAP group communication PUT request, and handling the potential multitude of (unicast) responses.

The base protocol aspects of sending CoAP requests on top of IP multicast, and processing the (unicast IP) responses are given in Section 8 of [RFC7252]. To provide a more complete CoAP group communication functionality, this specification introduces new CoAP protocol processing functionality (e.g., new rules for re-use of

Token values, request suppression, and proxy operation) and a new management interface for RESTful group membership configuration.

CoAP group communication will run in Any Source Multicast (ASM) mode [RFC5110] of IP multicast operation. This means that there is no restriction on the source node which sends (originates) the CoAP messages to the IP multicast group. For example, the source node may be part of the IP multicast group or not. Also, there is no restriction on the number of source nodes.

While Section 9.1 of [RFC7252] supports various modes of DTLS-based security for CoAP over unicast IP, it does not specify any security modes for CoAP over IP multicast. That is, [RFC7252] assumes that CoAP over IP multicast is not encrypted, nor authenticated, nor access controlled. This document assumes the same security model (see Section 5.1). However, there are several promising security approaches being developed that should be considered in the future for protecting CoAP group communications (see Section 5.3.3).

### 1.3. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

Note that this document refers back to other RFCs, and especially [RFC7252], to help explain overall CoAP group communication features. However use of [RFC2119] key words is reserved for new CoAP functionality introduced by this specification.

This document assumes readers are familiar with the terms and concepts that are used in [RFC7252]. In addition, this document defines the following terminology:

#### Group Communication

A source node sends a single application layer (e.g., CoAP) message which is delivered to multiple destination nodes, where all destinations are identified to belong to a specific group. The source node itself may be part of the group. The underlying mechanisms for CoAP group communication are UDP/IP multicast for the requests, and unicast UDP/IP for the responses. The network involved may be a constrained network such as a low-power, lossy network.



#### Reliable Group Communication

A special case of group communication where for each destination node it is guaranteed that the node either 1) eventually receives the message sent by the source node, or 2) does not receive the message and the source node is notified of the non-reception event. An example of a reliable group communication protocol is [RFC5740].

#### Multicast

Sending a message to multiple destination nodes with one network invocation. There are various options to implement multicast including layer 2 (Media Access Control) and layer 3 (IP) mechanisms.

#### IP Multicast

A specific multicast approach based on the use of IP multicast addresses as defined in "IANA Guidelines for IPv4 Multicast Address Assignments" [RFC5771] and "IP Version 6 Addressing Architecture" [RFC4291]. A complete IP multicast solution may include support for managing group memberships, and IP multicast routing/forwarding (see Section 2.1).

#### Low power and Lossy Network (LLN)

A type of constrained IP network where devices are interconnected by low-power and lossy links. The links may be composed of one or more technologies such as IEEE 802.15.4, Bluetooth Low Energy (BLE), Digital Enhanced Cordless Telecommunication (DECT), and IEEE P1901.2 power-line communication.

## 2. Protocol Considerations

### 2.1. IP Multicast Background

IP multicast protocols have been evolving for decades, resulting in standards such as Protocol Independent Multicast - Sparse Mode (PIM-SM) [RFC4601]. IP multicast is very popular in specific deployments such as in enterprise networks (e.g., for video conferencing), smart home networks (e.g., Universal Plug and Play (UPnP)) and carrier IPTV deployments. The packet economy and minimal host complexity of IP multicast make it attractive for group communication in constrained environments.

To achieve IP multicast beyond link-local scope, an IP multicast routing or forwarding protocol needs to be active on IP routers. An example of a routing protocol specifically for LLNs is the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) (Section 12 of [RFC6550]) and an example of a forwarding protocol for LLNs is Multicast Protocol for Low power and Lossy Networks (MPL)

[I-D.ietf-roll-trickle-mcast]. RPL and MPL do not depend on each other; each can be used in isolation and both can be used in combination in a network. Finally, PIM-SM [RFC4601] is often used for multicast routing in traditional IP networks (i.e., networks that are not constrained).

IP multicast can also be run in a Link-Local (LL) scope. This means that there is no routing involved and an IP multicast message is only received over the link on which it was sent.

For a complete IP multicast solution, in addition to a routing/forwarding protocol, a "listener" protocol may be needed for the devices to subscribe to groups (see Section 4.2). Also, a multicast forwarding proxy node [RFC4605] may be required.

IP multicast is generally classified as an unreliable service in that packets are not guaranteed to be delivered to each and every member of the group. In other words, it cannot be directly used as a basis for "reliable group communication" as defined in Section 1.3. However, the level of reliability can be increased by employing a multicast protocol that performs periodic retransmissions as is done, for example, in MPL.

## 2.2. Group Definition and Naming

A CoAP group is defined as a set of CoAP endpoints, where each endpoint is configured to receive CoAP group communication requests that are sent to the group's associated IP multicast address. The individual response by each endpoint receiver to a CoAP group communication request is always sent back as unicast. An endpoint may be a member of multiple groups. Group membership of an endpoint may dynamically change over time.

All CoAP server nodes SHOULD join the "All CoAP Nodes" multicast group (Section 12.8 of [RFC7252]) by default to enable CoAP discovery. For IPv4, the address is 224.0.1.187 and for IPv6 a server node joins at least both the link-local scoped address FF02::FD and the site-local scoped address FF05::FD. IPv6 addresses of other scopes MAY be enabled.

A CoAP group URI has the scheme 'coap' and includes in the authority part either a group IP multicast address, or a hostname (e.g., Group Fully Qualified Domain Name (FQDN)) that can be resolved to the group IP multicast address. A group URI also contains an optional CoAP port number in the authority part. Group URIs follow the regular CoAP URI syntax (Section 6 of [RFC7252]).

Note: A group URI is needed to initiate CoAP group communications. For CoAP client implementations it is recommended to use the URI decomposition method of Section 6.4 of [RFC7252] in such way that, from a group URI, a CoAP group communication request is generated.

For sending nodes, it is recommended to use the IP multicast address literal in a group URI. (This is because DNS infrastructure may not be deployed in many constrained network deployments). However, in case a group hostname is used, it can be uniquely mapped to an IP multicast address via DNS resolution (if supported). Some examples of hierarchical group FQDN naming (and scoping) for a building control application are shown below:

| URI authority                           | Targeted group of nodes  |
|---|--|
| -----                                   | -----  |
| all.bldg6.example.com                   | "all nodes in building 6"  |
| all.west.bldg6.example.com              | "all nodes in west wing,<br>building 6"                          |
| all.floor1.west.bldg6.example.com       | "all nodes in floor 1,<br>west wing, building 6"                 |
| all.bu036.floor1.west.bldg6.example.com | "all nodes in office bu036,<br>floor1, west wing,<br>building 6" |

Similarly, if supported, reverse mapping (from IP multicast address to Group FQDN) is possible using the reverse DNS resolution technique ([RFC1033]). Reverse mapping is important, for example, in trouble shooting to translate IP multicast addresses back to human readable hostnames to show in a diagnostics user interface.

### 2.3. Port and URI Configuration

A CoAP server that is a member of a group listens for CoAP messages on the group's IP multicast address, usually on the CoAP default UDP port, 5683. If the group uses a specified non-default UDP port, be careful to ensure that all group members are configured to use that same port.

Different ports for the same IP multicast address are preferably not used to specify different CoAP groups. If disjoint groups share the same IP multicast address, then all the devices interested in one group will accept IP traffic also for the other disjoint groups, only to ultimately discard the traffic higher in their IP stack (based on UDP port discrimination).

CoAP group communication will not work if there is diversity in the authority port (e.g., different dynamic port addresses across the group) or if other parts of the group URI such as the path, or the

query, differ on different endpoints. Therefore, some measures must be present to ensure uniformity in port number and resource names/locations within a group. All CoAP group communication requests **MUST** be sent using a port number according to one of below options:

1. A pre-configured port number.
2. If the client is configured to use service discovery including URI and port discovery, it uses the port number obtained via a service discovery lookup operation for the targeted CoAP group.
3. Use the default CoAP UDP port (5683).

For a CoAP server node that supports resource discovery, the default port 5683 must be supported (Section 7.1 of [RFC7252]) for the "All CoAP Nodes" group. Regardless of the method of selecting the port number, the same port **MUST** be used across all CoAP servers in a group and across all CoAP clients performing the group requests.

All CoAP group communication requests **SHOULD** operate on group URI paths in one of the following ways:

1. Pre-configured group URI paths, if available. Implementers are free to define the paths as they see fit. However, note that [RFC7320] prescribes that a specification must not constrain, define structure or semantics for any path component. So for this reason, a pre-defined URI path is not specified in this document and also must not be provided in other specifications.
2. If the client is configured to use default CoRE resource discovery, it uses URI paths retrieved from a `"/.well-known/core"` lookup on a group member. The URI paths the client will use **MUST** be known to be available also in all other endpoints in the group. The URI path configuration mechanism on servers **MUST** ensure that these URIs (identified as being supported by the group) are configured on all group endpoints.
3. If the client is configured to use another form of service discovery, it uses group URI paths from an equivalent service discovery lookup which returns the resources supported by all group members.
4. If the client has received a group URI through a previous RESTful interaction with a trusted server it can use this URI in a CoAP group communication request. For example, a commissioning tool may instruct a sensor device in this way to which target group (group URI) it should report sensor events.

However the URI path is selected, the same path MUST be used across all CoAP servers in a group and all CoAP clients performing the group requests.

## 2.4. RESTful Methods

Group communication most often uses the idempotent CoAP methods GET and PUT. The idempotent method DELETE can also be used. The non-idempotent CoAP method POST may only be used for group communication if the resource being POSTed to has been designed to cope with the unreliable and lossy nature of IP multicast. For example, a client may re-send a multicast POST request for additional reliability. Some servers will receive the request two times while others may receive it only once. For idempotent methods all these servers will be in the same state, while for POST this is not guaranteed; so the resource POST operation must be specifically designed to take message loss into account.

## 2.5. Request and Response Model

All CoAP requests that are sent via IP multicast must be Non-confirmable (Section 8.1 of [RFC7252]). The Message ID in an IP multicast CoAP message is used for optional message de-duplication as detailed in Section 4.5 of [RFC7252].

A server optionally sends back a unicast response to the CoAP group communication request (e.g., response "2.05 Content" to a group GET request). The unicast responses received by the CoAP client may be a mixture of success (e.g., 2.05 Content) and failure (e.g., 4.04 Not Found) codes depending on the individual server processing results. Detailed processing rules for IP multicast request acceptance and unicast response suppression are given in Section 2.7.

A CoAP request sent over IP multicast and any unicast response it causes must take into account the congestion control rules defined in Section 2.8.

The CoAP client can distinguish the origin of multiple server responses by source IP address of the UDP message containing the CoAP response, or any other available unique identifier (e.g., contained in the CoAP payload). In case a CoAP client sent multiple group requests, the responses are as usual matched to a request using the CoAP Token.

For multicast CoAP requests there are additional constraints on the re-use of Token values, compared to the unicast case. In the unicast case, receiving a response effectively frees up its Token value for re-use since no more responses will follow. However, for multicast

CoAP the number of responses is not bounded a-priori. Therefore the reception of a response cannot be used as a trigger to "free up" a Token value for re-use. Re-using a Token value too early could lead to incorrect response/request matching in the client, which is a protocol error. Therefore the time between re-use of Token values used in multicast requests MUST be greater than:

$$\text{NON\_LIFETIME} + \text{MAX\_LATENCY} + \text{MAX\_SERVER\_RESPONSE\_DELAY}$$

where NON\_LIFETIME and MAX\_LATENCY are defined in Section 4.8 of [RFC7252]. MAX\_SERVER\_RESPONSE\_DELAY is defined here as the expected maximum response delay over all servers that the client can send a multicast request to. This delay includes the maximum Leisure time period as defined in Section 8.2 of [RFC7252]. The CoAP protocol does not define a time limit for the server response delay. Using the default CoAP protocol parameters, the Token re-use time MUST be greater than 250 seconds plus MAX\_SERVER\_RESPONSE\_DELAY. A preferred solution to meet this requirement is to generate a new unique Token for every multicast request, such that a Token value is never re-used. If a client has to re-use Token values for some reason, and also MAX\_SERVER\_RESPONSE\_DELAY is unknown, then using MAX\_SERVER\_RESPONSE\_DELAY = 250 seconds is a reasonable guideline. The time between Token re-uses is in that case set to a value greater than 500 seconds.

## 2.6. Membership Configuration

### 2.6.1. Background

#### 2.6.1.1. Member Discovery

CoAP Groups, and the membership of these groups, can be discovered via the lookup interfaces in the Resource Directory (RD) defined in [I-D.ietf-core-resource-directory]. This discovery interface is not required to invoke CoAP group communications. However, it is a potential complementary interface useful for overall management of CoAP groups. Other methods to discover groups (e.g., proprietary management systems) can also be used. An example of doing some of the RD based lookups is given in Section 3.6.

#### 2.6.1.2. Configuring Members

The group membership of a CoAP endpoint may be configured in one of the following ways. First, the group membership may be pre-configured before node deployment. Second, a node may be programmed to discover (query) its group membership using a specific service discovery means. Third, it may be configured by another node (e.g., a commissioning device).

In the first case, the pre-configured group information may be either an IP multicast address or a hostname (FQDN) which is resolved later (during operation) to an IP multicast address by the endpoint using DNS (if supported).

For the second case, a CoAP endpoint may look up its group membership using techniques such as DNS-SD and Resource Directory [I-D.ietf-core-resource-directory].

In the third case, typical in scenarios such as building control, a dynamic commissioning tool determines to which group(s) a sensor or actuator node belongs, and writes this information to the node, which can subsequently join the correct IP multicast group(s) on its network interface. The information written per group may again be an IP multicast address or a hostname.

#### 2.6.2. Membership Configuration RESTful Interface

To achieve better interoperability between endpoints from different manufacturers, an OPTIONAL CoAP membership configuration RESTful interface for configuring endpoints with relevant group information is described here. This interface provides a solution for the third case mentioned above. To access this interface a client will use unicast CoAP methods (GET/PUT/POST/DELETE). This interface is a method of configuring group information in individual endpoints.

Also, a form of authorization (preferably making use of unicast DTLS-secured CoAP of Section 9.1 of [RFC7252]) should be used such that only authorized controllers are allowed by an endpoint to configure its group membership.

It is important to note that other approaches may be used to configure CoAP endpoints with relevant group information. These alternative approaches may support a subset or super-set of the membership configuration RESTful interface described in this document. For example, a simple interface to just read the endpoint group information may be implemented via a classical Management Information Base (MIB) approach (e.g., following approach of [RFC3433]).

##### 2.6.2.1. CoAP-Group Resource Type and Media Type

CoAP endpoints implementing the membership configuration RESTful interface MUST support the CoAP group configuration Internet Media Type "application/coap-group+json" (Section 6.2).

A resource offering this representation can be annotated for direct discovery [RFC6690] using the resource type (rt) "core.gp" where "gp"

is shorthand for "group" (Section 6.1). An authorized client uses this media type to query/manage group membership of a CoAP endpoint as defined in the following subsections.

The group configuration resource and its sub-resources have a JavaScript Object Notation (JSON) based content format (as indicated by the "application/coap-group+json" media type). The resource includes zero or more group membership JSON objects [RFC7159] in a format as defined in Section 2.6.2.4. A group membership JSON object contains one or more key/value pairs as defined below, and represents a single IP multicast group membership for the CoAP endpoint. Each key/value pair is encoded as a member of the JSON object, where the key is the member name and the value is the member's value.

Examples of four different group membership objects are:

```
{ "n": "All-Devices.floor1.west.bldg6.example.com",  
  "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }  
  
{ "n": "sensors.floor2.east.bldg6.example.com" }  
  
{ "n": "coap-test",  
  "a": "224.0.1.187:56789" }  
  
{ "a": "[ff15::c0a7:15:c001]" }
```

The OPTIONAL "n" key/value pair stands for "name" and identifies the group with a hostname (and optionally the port number), for example, a FQDN. The OPTIONAL "a" key/value pair specifies the IP multicast address (and optionally the port number) of the group. It contains an IPv4 address (in dotted decimal notation) or an IPv6 address. The following ABNF rule can be used for parsing the address, referring to the definitions in Section 3.2.2 of [RFC3986] which are also used in the base CoAP protocol (Section 6 of [RFC7252]).

```
group-address = IPv4address [ ":" port ]  
               / "[" IPv6address "]" [ ":" port ]
```

In any group membership object, if the IP address is known when the object is created, it is included in the "a" key/value pair. If the "a" value cannot be provided, the "n" value MUST be included, containing a valid hostname with optional port number that can be translated to an IP multicast address via DNS.

```
group-name = host [ ":" port ]
```

If the port number is not provided, then the endpoint will attempt to look up the port number from DNS if it supports a method to do this.



The possible DNS methods include DNS-SRV [RFC2782] or DNS-SD [RFC6763]. If port lookup is not supported or not provided by DNS, the default CoAP port (5683) is assumed.

After any change on a Group configuration resource, the endpoint MUST effect registration/de-registration from the corresponding IP multicast group(s) by making use of APIs such as IPV6\_RECVPKTINFO [RFC3542].

#### 2.6.2.2. Creating a new multicast group membership (POST)

Method: POST  
URI Template: /{+gp}  
Location-URI Template: /{+gp}/{index}  
URI Template Variables:  
    gp - Group Configuration Function Set path (mandatory).  
    index - Group index. Index MUST be a string of maximum two (2) alphanumeric ASCII characters (case insensitive). It MUST be locally unique to the endpoint server.  
    It indexes the particular endpoint's list of group memberships.

#### Example:

```
Req: POST /coap-group
    Content-Format: application/coap-group+json
    { "n": "All-Devices.floor1.west.bldg6.example.com",
      "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }
Res: 2.01 Created
    Location-Path: /coap-group/12
```

For the 'gp' variable it is recommended to use the path "coap-group" by default. The "a" key/value pair is always used if it is given. The "n" pair is only used when there is no "a" pair. If only the "n" pair is given, the CoAP endpoint performs DNS resolution to obtain the IP multicast address from the hostname in the "n" pair. If DNS resolution is not successful, then the endpoint does not attempt joining or listening to any multicast group for this case since the IP multicast address is unknown.

After any change on a Group configuration resource, the endpoint MUST effect registration/de-registration from the corresponding IP multicast group(s) by making use of APIs such as IPV6\_RECVPKTINFO [RFC3542]. When a POST payload contains in "a" an IP multicast address to which the endpoint is already subscribed, no change to that subscription is needed.

#### 2.6.2.3. Deleting a single group membership (DELETE)

Method: DELETE  
URI Template: {+location}  
URI Template Variables:  
    location - The Location-Path returned by the CoAP server as a result  
            of a successful group creation.

Example:

Req: DELETE /coap-group/12  
Res: 2.02 Deleted

#### 2.6.2.4. Reading all group memberships at once (GET)

A (unicast) GET on the CoAP-group resource returns a JSON object containing multiple keys and values. The keys (member names) are group indices and the values (member values) are the corresponding group membership objects. Each group membership object describes one IP multicast group membership. If no group memberships are configured then an empty JSON object is returned.

Method: GET

URI Template: /{+gp}

URI Template Variables:

gp - see Section 2.6.2.2

Example:

Req: GET /coap-group  
Res: 2.05 Content  
    Content-Format: application/coap-group+json  
    { "8" :{ "a": "[ff15::4200:f7fe:ed37:14ca]" },  
      "11":{ "n": "sensors.floor1.west.bldg6.example.com",  
            "a": "[ff15::4200:f7fe:ed37:25cb]" },  
      "12":{ "n": "All-Devices.floor1.west.bldg6.example.com",  
            "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }  
    }

Note: the returned IPv6 address string will represent the same IPv6 address that was originally submitted in group membership creation, though it might be a different string because of different choices in IPv6 string representation formatting that may be allowed for the same address (see [RFC5952]).

#### 2.6.2.5. Reading a single group membership (GET)

Similar to Section 2.6.2.4 but only a single group membership is read. If the requested group index does not exist then a 4.04 Not Found response is returned.

Method: GET

URI Template 1: {+location}

URI Template 2: /{+gp}/{index}

URI Template Variables:

location - see Section 2.6.2.3

gp, index - see Section 2.6.2.2

Example:

Req: GET /coap-group/12

Res: 2.05 Content

Content-Format: application/coap-group+json

```
{ "n": "All-Devices.floor1.west.bldg6.example.com",  
  "a": "[ff15::4200:f7fe:ed37:abcd]:4567" }
```

#### 2.6.2.6. Creating/updating all group memberships at once (PUT)

A (unicast) PUT with a group configuration media type as payload will replace all current group memberships in the endpoint with the new ones defined in the PUT request. This operation **MUST** only be used to delete or update group membership objects for which the CoAP client, invoking this operation, is responsible. The responsibility is based on application level knowledge. For example, a commissioning tool will be responsible for any group membership objects that it created.

Method: PUT

URI Template: /{+gp}

URI Template Variables:

gp - see Section 2.6.2.2

Example: (replacing all existing group memberships with two new group memberships)

```
Req: PUT /coap-group
    Content-Format: application/coap-group+json
    { "1":{ "a": "[ff15::4200:f7fe:ed37:1234]" },
      "2":{ "a": "[ff15::4200:f7fe:ed37:5678]" }
    }
Res: 2.04 Changed
```

Example: (clearing all group memberships at once)

```
Req: PUT /coap-group
    Content-Format: application/coap-group+json
    {}
Res: 2.04 Changed
```

After a successful PUT on the Group configuration resource, the endpoint MUST effect registration to any new IP multicast group(s) and de-registration from any previous IP multicast group(s), i.e., not any more present in the new memberships. An API such as IPV6\_RECVPKTINFO [RFC3542] should be used for this purpose. Also it MUST take into account the group indices present in the new resource during the generation of any new unique group indices in the future.

#### 2.6.2.7. Updating a single group membership (PUT)

A (unicast) PUT with a group membership JSON object will replace an existing group membership in the endpoint with the new one defined in the PUT request. This can be used to update the group membership.

Method: PUT

URI Template 1: {+location}

URI Template 2: /{+gp}/{index}

URI Template Variables:

location - see Section 2.6.2.3

gp, index - see Section 2.6.2.2

Example: (group name and IP multicast port change)

```
Req: PUT /coap-group/l2
    Content-Format: application/coap-group+json
    { "n": "All-My-Devices.floor1.west.bldg6.example.com",
      "a": "[ff15::4200:f7fe:ed37:abcd]" }
Res: 2.04 Changed
```

After a successful PUT on the Group configuration resource, the endpoint MUST effect registration to any new IP multicast group(s) and de-registration from any previous IP multicast group(s), i.e., not any more present in the new membership. An API such as IPV6\_RECVPKTINFO [RFC3542] should be used for this purpose.

## 2.7. Request Acceptance and Response Suppression Rules

CoRE Link Format [RFC6690], and Section 8 of CoAP [RFC7252] define behaviors for:

1. IP multicast request acceptance - in which cases a CoAP request is accepted and executed, and when not.
2. IP multicast response suppression - in which cases the CoAP response to an already-executed request is returned to the requesting endpoint, and when not.

A CoAP response differs from a CoAP ACK; ACKs are never sent by servers in response to an IP multicast CoAP request. This section first summarizes these behaviors and then presents additional guidelines for response suppression. Also a number of IP multicast example applications are given to illustrate the overall approach.

To apply any rules for request and/or response suppression, a CoAP server must be aware that an incoming request arrived via IP multicast by making use of APIs such as IPV6\_RECVPKTINFO [RFC3542].

For IP multicast request acceptance, the behaviors are:

- o A server should not accept an IP multicast request that cannot be "authenticated" in some way (i.e, cryptographically or by some multicast boundary limiting the potential sources) (Section 11.3 of [RFC7252]. See Section 5.3 for examples of multicast boundary limiting methods.
- o A server should not accept an IP multicast discovery request with a query string (as defined in CoRE Link Format [RFC6690]) if filtering ([RFC6690]) is not supported by the server.
- o A server should not accept an IP multicast request that acts on a specific resource for which IP multicast support is not required. (Note that for the resource "/.well-known/core", IP multicast support is required if "multicast resource discovery" is supported as specified in Section 1.2.1 of [RFC6690]). Implementers are advised to disable IP multicast support by default on any other resource, until explicitly enabled by an application or by configuration.)

- o Otherwise accept the IP multicast request.

For IP multicast response suppression, the behaviors are:

- o A server should not respond to an IP multicast discovery request if the filter specified by the request's query string does not match.
- o A server may choose not to respond to an IP multicast request, if there's nothing useful to respond (e.g., error or empty response).

The above response suppression behaviors are complemented by the following guidelines. CoAP servers should implement configurable response suppression, enabling at least the following options per resource that supports IP multicast requests:

- o Suppression of all 2.xx success responses;
- o Suppression of all 4.xx client errors;
- o Suppression of all 5.xx server errors;
- o Suppression of all 2.05 responses with empty payload.

A number of CoAP group communication example applications are given below to illustrate how to make use of response suppression:

- o CoAP resource discovery: Suppress 2.05 responses with empty payload and all 4.xx and 5.xx errors.
- o Lighting control: Suppress all 2.xx responses after a lighting change command.
- o Update configuration data in a group of devices using group communication PUT: No suppression at all. The client uses collected responses to identify which group members did not receive the new configuration; then attempts using CoAP CON unicast to update those specific group members. Note that in this case the client implements a "reliable group communication" (as defined in Section 1.3) function using additional, non-standardized functions above the CoAP layer.
- o IP multicast firmware update by sending blocks of data: Suppress all 2.xx and 5.xx responses. After having sent all IP multicast blocks, the client checks each endpoint by unicast to identify which data blocks are still missing in each endpoint.

- o Conditional reporting for a group (e.g., sensors) based on a group URI query: Suppress all 2.05 responses with empty payload (i.e., if a query produces no matching results).

## 2.8. Congestion Control

CoAP group communication requests may result in a multitude of responses from different nodes, potentially causing congestion. Therefore both the sending of IP multicast requests, and the sending of the unicast CoAP responses to these multicast requests should be conservatively controlled.

CoAP [RFC7252] reduces IP multicast-specific congestion risks through the following measures:

- o A server may choose not to respond to an IP multicast request if there's nothing useful to respond (e.g., error or empty response)(Section 8.2 [RFC7252]). See Section 2.7 for more detailed guidelines on response suppression.
- o A server should limit the support for IP multicast requests to specific resources where multicast operation is required (Section 11.3 of [RFC7252]).
- o An IP multicast request must be Non-confirmable (Section 8.1 of [RFC7252]).
- o A response to an IP multicast request should be Non-confirmable (Section 5.2.3 of [RFC7252]).
- o A server does not respond immediately to an IP multicast request, and should first wait for a time that is randomly picked within a predetermined time interval called the Leisure (Section 8.2 [RFC7252]).

Additional guidelines to reduce congestion risks defined in this document are:

- o A server in an LLN should only support group communication GET for resources that are small. For example, the payload of the response is limited to approximately 5% of the IP Maximum Transmit Unit (MTU) size so it fits into a single link-layer frame in case 6LoWPAN (see Section 4 of [RFC4944]) is used.
- o A server can minimize the payload length in response to a group communication GET on "/.well-known/core" by using hierarchy in arranging link descriptions for the response. An example of this is given in Section 5 of [RFC6690].

- o A server can also minimize the payload length of a response to a group communication GET (e.g., on `"/.well-known/core"`) using CoAP blockwise transfers [I-D.ietf-core-block], returning only a first block of the CoRE Link Format description. For this reason, a CoAP client sending an IP multicast CoAP request to `"/.well-known/core"` should support core-block.
- o A client should use CoAP group communication with the smallest possible IP multicast scope that fulfills the application needs. As an example, site-local scope is always preferred over global scope IP multicast if this fulfills the application needs.

More guidelines specific to use of CoAP in 6LoWPAN networks [RFC4919] are given in Section 4.5.

## 2.9. Proxy Operation

CoAP (Section 5.7.2 of [RFC7252]) allows a client to request a forward-proxy to process its CoAP request. For this purpose the client either specifies the request group URI as a string in the Proxy-URI option, or it specifies the Proxy-Scheme option with the group URI constructed from the usual Uri-\* options. This approach works well for unicast requests. However, there are certain issues and limitations of processing the (unicast) responses to a CoAP group communication request made in this manner through a proxy.

A proxy may buffer all the individual (unicast) responses to a CoAP group communication request and then send back only a single (aggregated) response to the client. However there are some issues with this aggregation approach:

- o Aggregation of (unicast) responses to a CoAP group communication request in a proxy is difficult. This is because the proxy does not know how many members there are in the group, or how many group members will actually respond. Also the proxy does not know how long to wait before deciding to send back the aggregated response to the client.
- o There is no default format defined in CoAP for aggregation of multiple responses into a single response.

Alternatively, if a proxy follows directly the specification for a CoAP Proxy (Section 5.7.2 of [RFC7252]), the proxy would simply forward all the individual (unicast) responses to a CoAP group communication request to the client (i.e., no aggregation). There are also issues with this approach:



- o The client may be confused as it may not have known that the Proxy-URI contained a group URI target. That is, the client may be expecting only one (unicast) response but instead receives multiple (unicast) responses potentially leading to fault conditions in the application.
- o Each individual CoAP response will appear to originate (IP Source address) from the CoAP Proxy, and not from the server that produced the response. This makes it impossible for the client to identify the server that produced each response.

Due to above issues, a CoAP Proxy SHOULD NOT support processing an IP multicast CoAP request but rather return a 501 (Not Implemented) response in such case. The exception case here (i.e., to process it) is allowed if all the following conditions are met:

- o The CoAP Proxy MUST be explicitly configured (whitelist) to allow proxied IP multicast requests by specific client(s).
- o The proxy SHOULD return individual (unicast) CoAP responses to the client (i.e., not aggregated). The exception case here occurs when a (future) standardized aggregation format is being used.
- o It MUST be known to the person/entity doing the configuration of the proxy, or otherwise verified in some way, that the client configured in the whitelist supports receiving multiple responses to a proxied unicast CoAP request.

## 2.10. Exceptions

CoAP group communication using IP multicast offers improved network efficiency and latency among other benefits. However, group communication may not always be implementable in a given network. The primary reason for this will be that IP multicast is not (fully) supported in the network.

For example, if only the RPL protocol [RFC6550] is used in a network with its optional multicast support disabled, there will be no IP multicast routing at all. The only multicast that works in this case is link-local IPv6 multicast. This implies that any CoAP group communication request will be delivered to nodes on the local link only, regardless of the scope value used in the IPv6 destination address.

CoAP Observe [I-D.ietf-core-observe] is a feature for a client to "observe" resources (i.e. to retrieve a representation of a resource and keep this representation updated by the server over a period of

time). CoAP Observe does not support a group communication mode. CoAP Observe only supports a unicast mode of operation.

### 3. Use Cases and Corresponding Protocol Flows

#### 3.1. Introduction

The use of CoAP group communication is shown in the context of the following two use cases and corresponding protocol flows:

- o Discovery of RD [I-D.ietf-core-resource-directory]: discovering the local CoAP RD which contains links to resources stored on other CoAP servers [RFC6690].
- o Lighting Control: synchronous operation of a group of IPv6-connected lights (e.g., 6LoWPAN [RFC4944] lights).

#### 3.2. Network Configuration

To illustrate the use cases we define two IPv6 network configurations. Both are based on the topology as shown in Figure 1. The two configurations using this topology are:

1. Subnets are 6LoWPAN networks; the routers Rtr-1 and Rtr-2 are 6LoWPAN Border Routers (6LBRs, [RFC6775]).
2. Subnets are Ethernet links; the routers Rtr-1 and Rtr-2 are multicast-capable Ethernet routers.

Both configurations are further specified by the following:

- o A large room (Room-A) with three lights (Light-1, Light-2, Light-3) controlled by a Light Switch. The devices are organized into two subnets. In reality, there could be more lights (up to several hundreds) but, for clarity, only three are shown.
- o Light-1 and the Light Switch are connected to a router (Rtr-1).
- o Light-2 and the Light-3 are connected to another router (Rtr-2).
- o The routers are connected to an IPv6 network backbone which is also multicast enabled. In the general case, this means the network backbone and Rtr-1/Rtr-2 support a PIM based multicast routing protocol, and Multicast Listener Discovery (MLD) for forming groups.
- o A CoAP RD is connected to the network backbone.

- o The DNS server is optional. If the server is there (connected to the network backbone) then certain DNS based features are available (e.g., DNS resolution of hostname to IP multicast address). If the DNS server is not there, then different provisioning of the network is required (e.g., IP multicast addresses are hard-coded into devices, or manually configured, or obtained via a service discovery method).
- o A Controller (CoAP client) is connected to the backbone, which is able to control various building functions including lighting.

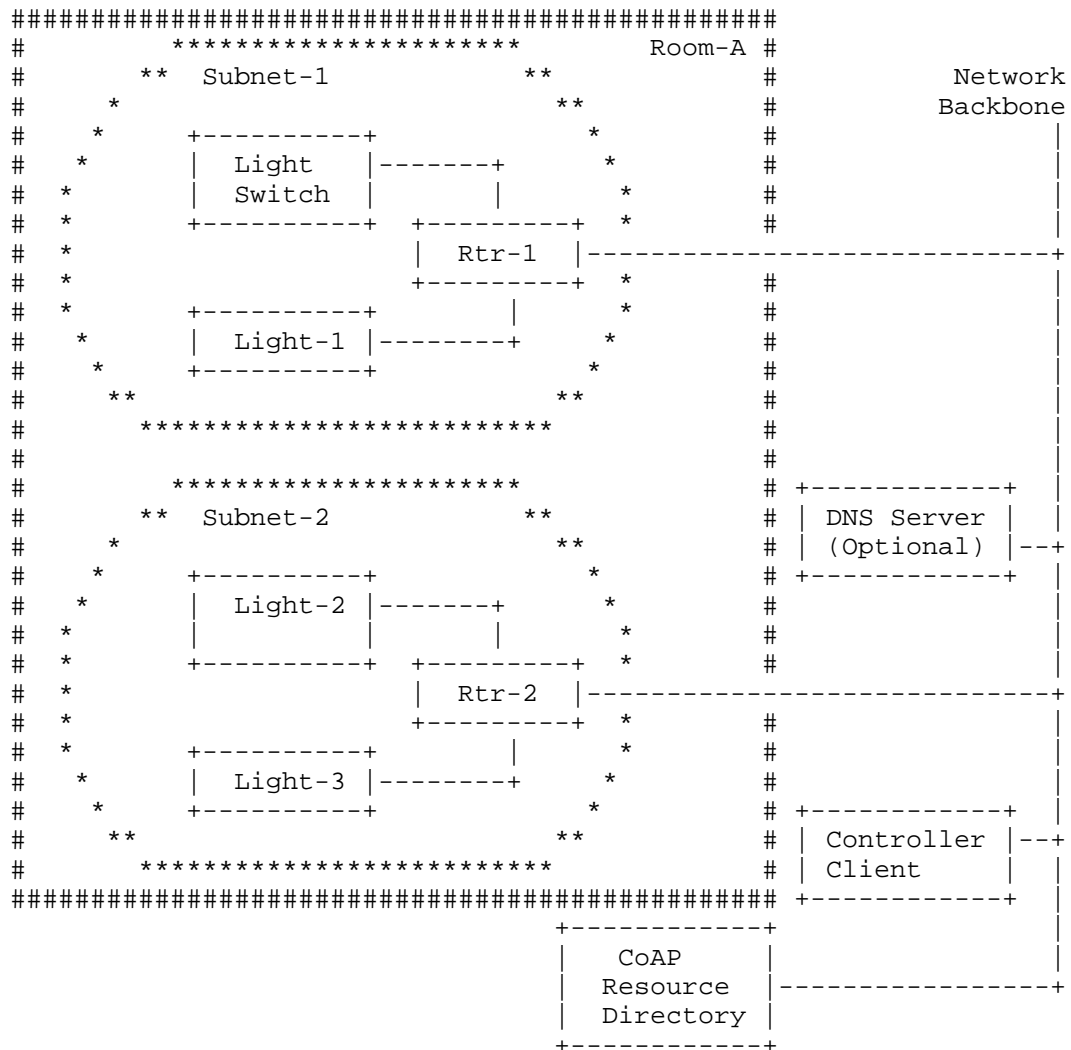


Figure 1: Network Topology of a Large Room (Room-A)

### 3.3. Discovery of Resource Directory

The protocol flow for discovery of the CoAP RD for the given network (of Figure 1) is shown in Figure 2:

- o Light-2 is installed and powered on for the first time.

- o Light-2 will then search for the local CoAP RD by sending out a group communication GET request (with the `"/.well-known/core?rt=core.rd"` request URI) to the site-local "All CoAP Nodes" multicast address (`FF05:::FD`).
- o This multicast message will then go to each node in subnet-2. Rtr-2 will then forward it into to the Network Backbone where it will be received by the CoAP RD. All other nodes in subnet-2 will ignore the group communication GET request because it is qualified by the query string `"?rt=core.rd"` (which indicates it should only be processed by the endpoint if it contains a resource of type `"core.rd"`).
- o The CoAP RD will then send back a unicast response containing the requested content, which is a CoRE Link Format representation of a resource of type `"core.rd"`.
- o Note that the flow is shown only for Light-2 for clarity. Similar flows will happen for Light-1, Light-3 and the Light Switch when they are first installed.

The CoAP RD may also be discovered by other means such as by assuming a default location (e.g., on a 6LBR), using DHCP, anycast address, etc. However, these approaches do not invoke CoAP group communication so are not further discussed here. (See [I-D.ietf-core-resource-directory] for more details).

For other discovery use cases such as discovering local CoAP servers, services or resources, CoAP group communication can be used in a similar fashion as in the above use case. For example, Link-Local (LL), admin-local or site-local scoped discovery can be done this way.

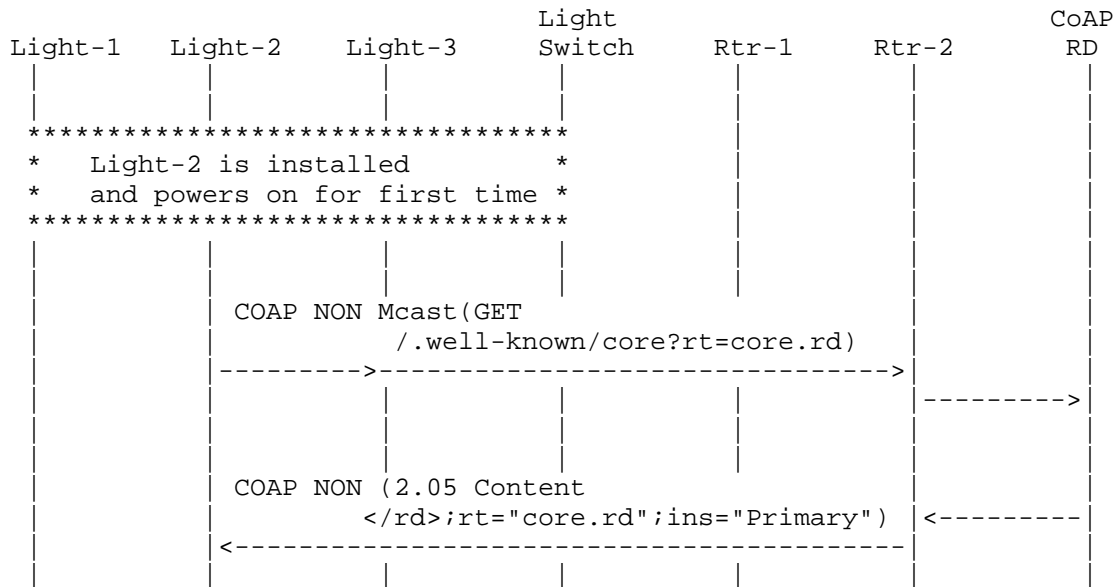


Figure 2: Resource Directory Discovery via Multicast Request

### 3.4. Lighting Control

The protocol flow for a building automation lighting control scenario for the network (Figure 1) is shown in Figure 3. The network is assumed to be in a 6LoWPAN configuration. Also, it is assumed that the CoAP servers in each Light are configured to suppress CoAP responses for any IP multicast CoAP requests related to lighting control. (See Section 2.7 for more details on response suppression by a server.)

In addition, Figure 4 shows a protocol flow example for the case that servers do respond to a lighting control IP multicast request with (unicast) CoAP NON responses. There are two success responses and one 5.00 error response. In this particular case, the Light Switch does not check that all Lights in the group received the IP multicast request by examining the responses. This is because the Light Switch is not configured with an exhaustive list of the IP addresses of all Lights belonging to the group. However, based on received error responses it could take additional action such as logging a fault or alerting the user via its LCD display. In case a CoAP message is delivered multiple times to a Light, the subsequent CoAP messages can be filtered out as duplicates, based on the CoAP Message ID.

Reliability of IP multicast is not guaranteed. Therefore, one or more lights in the group may not have received the CoAP control request due to packet loss. In this use case there is no detection nor correction of such situations: the application layer expects that the IP multicast forwarding/routing will be of sufficient quality to provide on average a very high probability of packet delivery to all CoAP endpoints in an IP multicast group. An example protocol to accomplish this using randomized retransmission is the MPL forwarding protocol for LLNs [I-D.ietf-roll-trickle-mcast].

We assume the following steps have already occurred before the illustrated flows:

- 1) Startup phase: 6LoWPANs are formed. IPv6 addresses assigned to all devices. The CoAP network is formed.

- 2) Network configuration (application-independent): 6LBRs are configured with IP multicast addresses, or address blocks, to filter out or to pass through to/from the 6LoWPAN.

- 3a) Commissioning phase (application-related): The IP multicast address of the group (Room-A-Lights) has been configured in all the Lights and in the Light Switch.

- 3b) As an alternative to the previous step, when a DNS server is available, the Light Switch and/or the Lights have been configured with a group hostname which each nodes resolves to the above IP multicast address of the group.

Note for the Commissioning phase: the switch's 6LoWPAN/CoAP software stack supports sending unicast, multicast or proxied unicast CoAP requests, including processing of the multiple responses that may be generated by an IP multicast CoAP request.

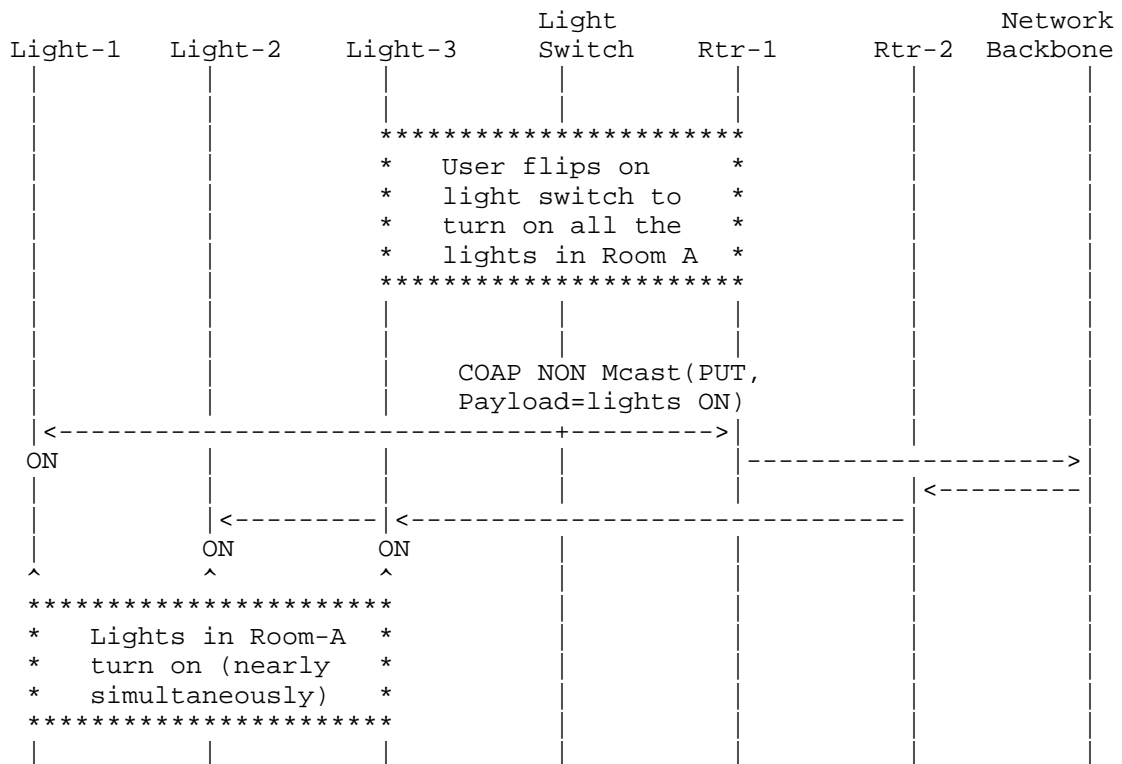


Figure 3: Light Switch Sends Multicast Control Message



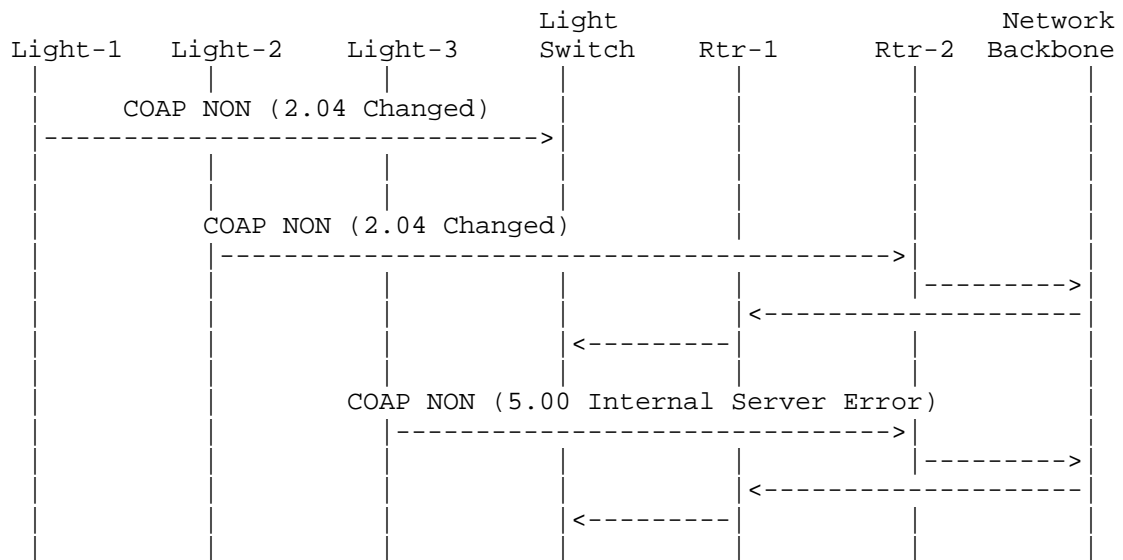


Figure 4: Lights (Optionally) Respond to Multicast CoAP Request

Another, but similar, lighting control use case is shown in Figure 5. In this case a controller connected to the Network Backbone sends a CoAP group communication request to turn on all lights in Room-A. Every Light sends back a CoAP response to the Controller after being turned on.

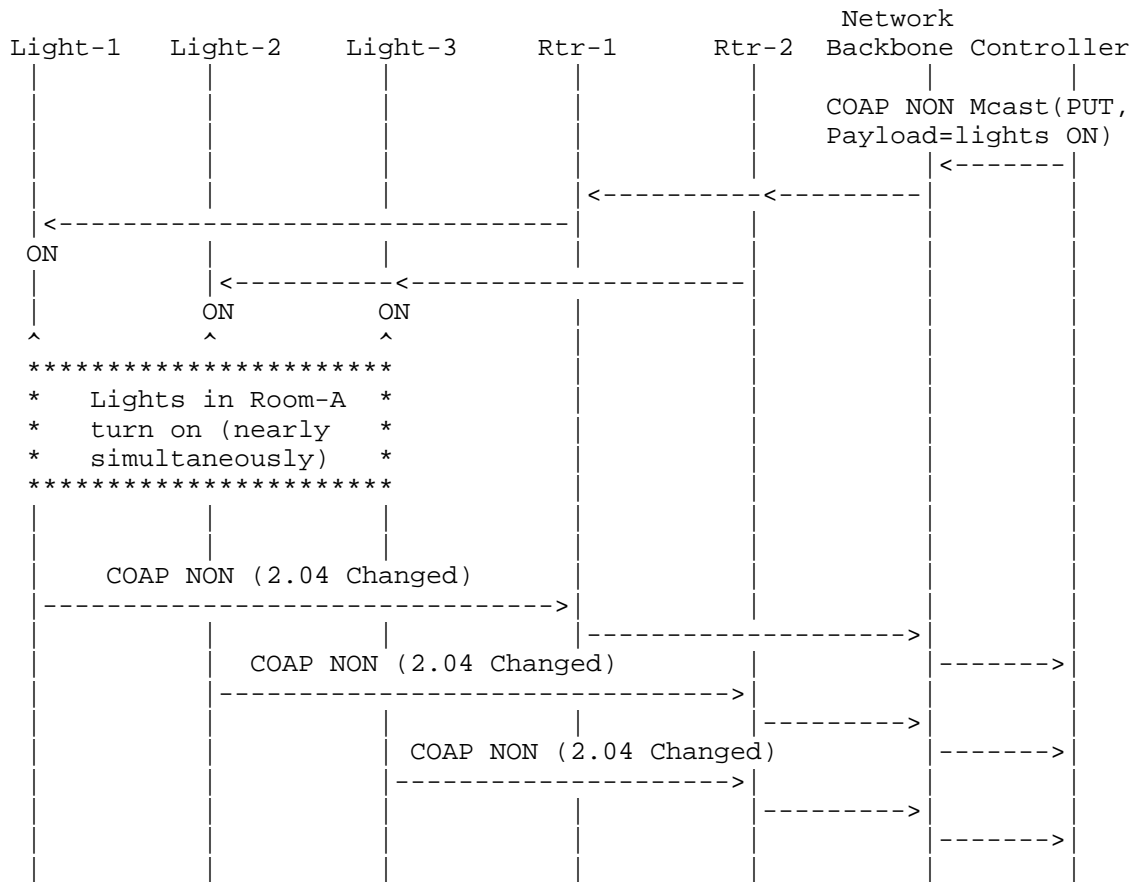


Figure 5: Controller On Backbone Sends Multicast Control Message

### 3.5. Lighting Control in MLD Enabled Network

The use case of previous section can also apply in networks where nodes support the MLD protocol [RFC3810]. The Lights then take on the role of MLDv2 listener and the routers (Rtr-1, Rtr-2) are MLDv2 Routers. In the Ethernet based network configuration, MLD may be available on all involved network interfaces. Use of MLD in the 6LoWPAN based configuration is also possible, but requires MLD support in all nodes in the 6LoWPAN. In current 6LoWPAN implementations, MLD is however not supported.

The resulting protocol flow is shown in Figure 6. This flow is executed after the commissioning phase, as soon as Lights are configured with a group address to listen to. The (unicast) MLD

Reports may require periodic refresh activity as specified by the MLD protocol. In the figure, LL denotes Link Local communication.

After the shown sequence of MLD Report messages has been executed, both Rtr-1 and Rtr-2 are automatically configured to forward IP multicast traffic destined to Room-A-Lights onto their connected subnet. Hence, no manual Network Configuration of routers, as previously indicated in Section 3.4, is needed anymore.

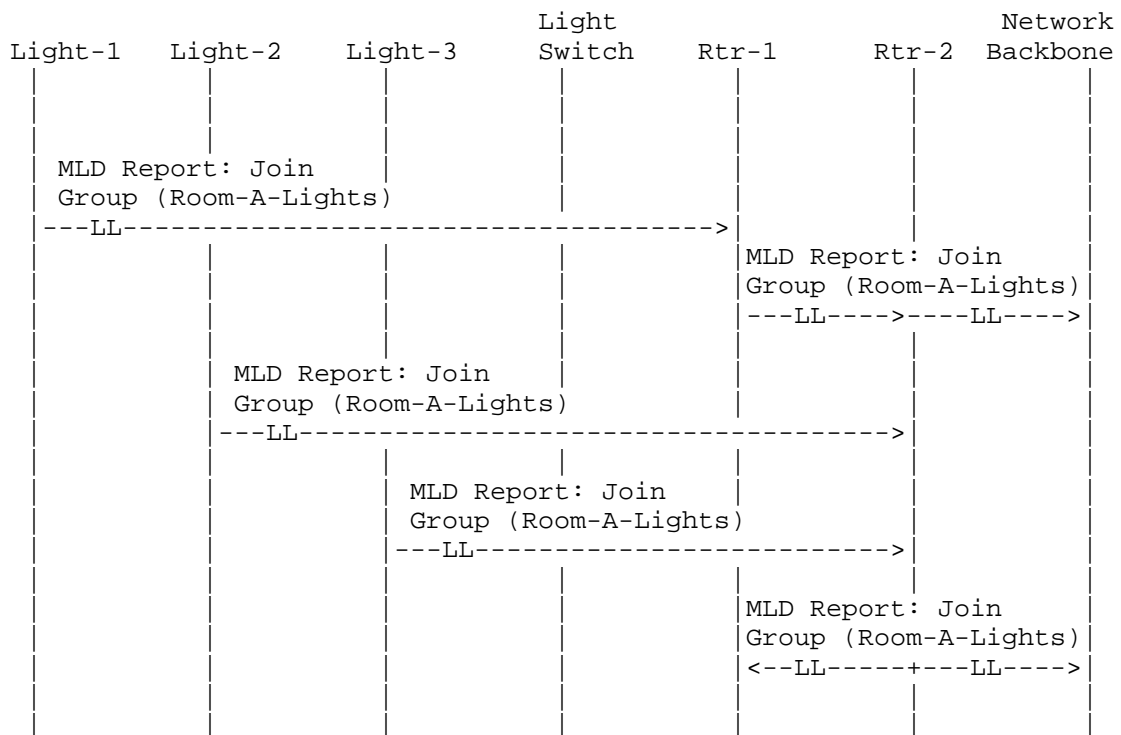


Figure 6: Joining Lighting Groups Using MLD

### 3.6. Commissioning the Network Based On Resource Directory

This section outlines how devices in the lighting use case (both Switches and Lights) can be commissioned, making use of Resource Directory [I-D.ietf-core-resource-directory] and its group configuration feature.

Once the Resource Directory (RD) is discovered, the Switches and Lights need to be discovered and their groups need to be defined.

For the commissioning of these devices, a commissioning tool can be used that defines the entries in the RD. The commissioning tool has the authority to change the contents of the RD and the Light/Switch nodes. DTLS-based unicast security is used by the commissioning tool to modify operational data in RD, Switches and Lights.

In our particular use case, a group of three lights is defined with one IP multicast address and hostname:

```
"Room-A-Lights.floor1.west.bldg6.example.com"
```

The commissioning tool has a list of the three lights and the associated IP multicast address. For each light in the list the tool learns the IP address of the light and instructs the RD with three (unicast) POST commands to store the endpoints associated with the three lights as prescribed by the RD specification [I-D.ietf-core-resource-directory]. Finally the commissioning tool defines the group in the RD to contain these three endpoints. Also the commissioning tool writes the IP multicast address in the Light endpoints with, for example, the (unicast) POST command discussed in Section 2.6.2.2.

The light switch can discover the group in RD and thus learn the IP multicast address of the group. The light switch will use this address to send CoAP group communication requests to the members of the group. When the message arrives the Lights should recognize the IP multicast address and accept the message.

#### 4. Deployment Guidelines

This section provides guidelines how IP multicast based CoAP group communication can be deployed in various network configurations.

##### 4.1. Target Network Topologies

CoAP group communication can be deployed in various network topologies. First, the target network may be a traditional IP network, or a LLN such as a 6LoWPAN network, or consist of mixed traditional/constrained network segments. Second, it may be a single subnet only or multi-subnet; e.g., multiple 6LoWPAN networks joined by a single backbone LAN. Third, a wireless network segment may have all its nodes reachable in a single IP hop (fully connected), or it may require multiple IP hops for some pairs of nodes to reach each other.

Each topology may pose different requirements on the configuration of routers and protocol(s), in order to enable efficient CoAP group

communication. To enable all the above target network topologies, an implementation of CoAP group communication needs to allow:

1. Routing/forwarding of IP multicast packets over multiple hops
2. Routing/forwarding of IP multicast packets over subnet boundaries between traditional and constrained (e.g., LLN) networks.

The remainder of this section discusses solutions to enable both features.

#### 4.2. Networks Using the MLD Protocol

CoAP nodes that are IP hosts (i.e., not IP routers) are generally unaware of the specific IP multicast routing/forwarding protocol being used. When such a host needs to join a specific (CoAP) multicast group, it requires a way to signal to IP multicast routers which IP multicast traffic it wants to receive.

The Multicast Listener Discovery (MLD) protocol [RFC3810] (see Appendix A) is the standard IPv6 method to achieve this; therefore this approach should be used on traditional IP networks. CoAP server nodes would then act in the role of MLD Multicast Address Listener.

The guidelines from [RFC6636] on tuning of MLD for mobile and wireless networks may be useful when implementing MLD in LLNs. However, on LLNs and 6LoWPAN networks the use of MLD may not be feasible at all due to constraints on code size, memory, or network capacity.

#### 4.3. Networks Using RPL Multicast Without MLD

It is assumed in this section that the MLD protocol is not implemented in a network, for example, due to resource constraints. The RPL routing protocol (see Section 12 of [RFC6550]) defines the advertisement of IP multicast destinations using Destination Advertisement Object (DAO) messages and routing of multicast IPv6 packets based on this. It requires the RPL Mode of Operation to be 3 (Storing Mode with multicast support).

Hence, RPL DAO can be used by CoAP nodes that are RPL Routers, or are RPL Leaf Nodes, to advertise IP multicast group membership to parent routers. Then, the RPL protocol is used to route IP multicast CoAP requests over multiple hops to the correct CoAP servers.

The same DAO mechanism can be used to convey IP multicast group membership information to an edge router (e.g., 6LBR), in case the edge router is also the root of the RPL DODAG. This is useful

because the edge router then learns which IP multicast traffic it needs to pass through from the backbone network into the LLN subnet. In 6LoWPAN networks, such selective "filtering" helps to avoid congestion of a 6LoWPAN subnet by IP multicast traffic from the traditional backbone IP network.

#### 4.4. Networks Using MPL Forwarding Without MLD

The MPL forwarding protocol [I-D.ietf-roll-trickle-mcast] can be used for propagation of IPv6 multicast packets to all MPL Forwarders within a predefined network domain, over multiple hops. MPL is designed to work in LLNs. In this section it is again assumed that Multicast Listener Discovery (MLD) is not implemented in the network, for example, due to resource limitations in an LLN.

The purpose of MPL is to let a predefined group of Forwarders collectively work towards the goal of distributing an IPv6 multicast packet throughout an MPL Domain. (A Forwarder node may be associated to multiple MPL Domains at the same time.) So it would appear there is no need for CoAP servers to advertise their multicast group membership, since any IP multicast packet that enters the MPL Domain is distributed to all MPL Forwarders without regard to what multicast addresses the individual nodes are listening to.

However, if an IP multicast request originates just outside the MPL Domain, the request will not be propagated by MPL. An example of such a case is the network topology of Figure 1 where the Subnets are 6LoWPAN subnets and per 6LoWPAN subnet one Realm-Local ([I-D.droms-6man-multicast-scopes]) MPL Domain is defined. The backbone network in this case is not part of any MPL Domain.

This situation can become a problem in building control use cases. For example, when the Controller Client needs to send a single IP multicast request to the group Room-A-Lights. By default, the request would be blocked by Rtr-1 and by Rtr-2, and not enter the Realm-Local MPL Domains associated to Subnet-1 and Subnet-2. The reason is that Rtr-1 and Rtr-2 do not have the knowledge that devices in Subnet-1/2 want to listen for IP packets destined to IP multicast group Room-A-Lights.

To solve the above issue, the following solutions could be applied:

1. Extend the MPL Domain. E.g. in above example, include the Network Backbone to be part of each of the two MPL Domains. Or in above example, create just a single MPL Domain that includes both 6LoWPAN subnets plus the backbone link, which is possible since MPL is not tied to a single link-layer technology.

2. Manual configuration of edge router(s) as MPL Seed(s) for specific IP multicast traffic. In the above example, this could be done through the following three steps: First, configure Rtr-1 and Rtr-2 to act as MLD Address Listeners for the Room-A-Lights IP multicast group. This step allows any (other) routers on the backbone to learn that at least one node on the backbone link is interested to receive any IP multicast traffic to Room-A-Lights. Second, configure both routers to "inject" any IP multicast packets destined to group Room-A-Lights into the (Realm-Local) MPL Domain that is associated to that router. Third, configure both routers to propagate any IPv6 multicast packets originating from within their associated MPL Domain to the backbone, if at least one node on the backbone has indicated interest to receive such IPv6 packets (for which MLD is used on the backbone).
3. Use an additional protocol/mechanism for injection of IP multicast traffic from outside an MPL Domain into that MPL Domain, based on IP multicast group subscriptions of Forwarders within the MPL Domain. Such protocol is currently not defined in [I-D.ietf-roll-trickle-mcast].

Concluding, MPL can be used directly in case all sources of IP multicast CoAP requests (CoAP clients) and also all the destinations (CoAP servers) are inside a single MPL Domain. Then, each source node acts as an MPL Seed. In all other cases, MPL can only be used with additional protocols and/or configuration on how IP multicast packets can be injected from outside into an MPL Domain.

#### 4.5. 6LoWPAN Specific Guidelines for the 6LBR

To support multi-subnet scenarios for CoAP group communication, it is recommended that a 6LoWPAN Border Router (6LBR) will act in an MLD Router role on the backbone link. If this is not possible then the 6LBR should be configured to act as an MLD Multicast Address Listener (see Appendix A) on the backbone link.

### 5. Security Considerations

This section describes the relevant security configuration for CoAP group communication using IP multicast. The threats to CoAP group communication are also identified and various approaches to mitigate these threats are summarized.

#### 5.1. Security Configuration

As defined in Sections 8.1 and 9.1 of [RFC7252], CoAP group communication based on IP multicast:

- o Will operate in CoAP NoSec (No Security) mode, until a future group security solution is developed (see also Section 5.3.3).
- o Will use the "coap" scheme. The "coaps" scheme should only be used when a future group security solution is developed (see also Section 5.3.3).

Essentially the above configuration means that there is currently no security at the CoAP layer for group communication. Therefore, for sensitive and mission critical applications (e.g., health monitoring systems, alarm monitoring systems) it is currently recommended to deploy CoAP group communication with an application-layer security mechanism (e.g, data object security) for improved security.

Application level security has many desirable properties including maintaining security properties while forwarding traffic through intermediaries (proxies). Application level security also tends to more cleanly separate security from the dynamics of group membership (e.g., the problem of distributing security keys across large groups with many members that come and go).

Without application-layer security, CoAP group communication should only be currently deployed in non-critical applications (e.g., read-only temperature sensors). Only when security solutions at the CoAP layer are mature enough (see Section 5.3.3) should CoAP group communication without application-layer security be considered for sensitive and mission-critical applications.

## 5.2. Threats

As noted above, there is currently no security at the CoAP layer for group communication. This is due to the fact that the current DTLS-based approach for CoAP is exclusively unicast oriented and does not support group security features such as group key exchange and group authentication. As a direct consequence of this, CoAP group communication is vulnerable to all attacks mentioned in Section 11 of [RFC7252] for IP multicast.

## 5.3. Threat Mitigation

Section 11 of [RFC7252] identifies various threat mitigation techniques for CoAP group communication. In addition to those guidelines, it is recommended that for sensitive data or safety-critical control, a combination of appropriate link-layer security and administrative control of IP multicast boundaries should be used. Some examples are given below.



### 5.3.1. WiFi Scenario

In a home automation scenario (using WiFi), the WiFi encryption should be enabled to prevent rogue nodes from joining. The Customer Premise Equipment (CPE) that enables access to the Internet should also have its IP multicast filters set so that it enforces multicast scope boundaries to isolate local multicast groups from the rest of the Internet (e.g., as per [RFC6092]). In addition, the scope of the IP multicast should be set to be site-local or smaller scope. For site-local scope, the CPE will be an appropriate multicast scope boundary point.

### 5.3.2. 6LoWPAN Scenario

In a building automation scenario, a particular room may have a single 6LoWPAN network with a single Edge Router (6LBR). Nodes on the subnet can use link-layer encryption to prevent rogue nodes from joining. The 6LBR can be configured so that it blocks any incoming (6LoWPAN-bound) IP multicast traffic. Another example topology could be a multi-subnet 6LoWPAN in a large conference room. In this case, the backbone can implement port authentication (IEEE 802.1X) to ensure only authorized devices can join the Ethernet backbone. The access router to this secured network segment can also be configured to block incoming IP multicast traffic.

### 5.3.3. Future Evolution

In the future, to further mitigate the threats, security enhancements need to be developed at IETF for group communications. This will allow introduction of a secure mode of CoAP group communication, and use of the "coaps" scheme for that purpose.

At the time of writing of this specification, there are various approaches being considered for security enhancements for group communications. Specifically, a lot of the current effort at IETF is geared towards developing a DTLS-based group communication. This is primarily motivated by the fact that the unicast CoAP security is DTLS-based (Section 9.1 of [RFC7252]). For example, [I-D.keoh-dice-multicast-security] proposes a DTLS-based IP multicast security. However, it is too early to conclude if this is the best approach. Alternatively, [I-D.mglt-dice-ipsec-for-application-payload] proposes an IPSec-based IP multicast security. This approach also needs further investigation and validation.

## 5.4. Monitoring Considerations

### 5.4.1. General Monitoring

CoAP group communication is meant to be used to control a set of related devices (e.g., simultaneously turn on all the lights in a room). This intrinsically exposes the group to some unique monitoring risks that solitary devices (i.e., devices not in a group) are not as vulnerable to. For example, assume an attacker is able to physically see a set of lights turn on in a room. Then the attacker can correlate a CoAP group communication message to that easily observable coordinated group action even if the contents of the message are encrypted by a future security solution (see Section 5.3.3). This will give the attacker side channel information to plan further attacks (e.g. by determining the members of the group then some network topology information may be deduced).

One mitigation to group communication monitoring risks that should be explored in the future is methods to de-correlate coordinated group actions. For example, if a CoAP group communication GET is sent to all the alarm sensors in a house, then their (unicast) responses should be as de-correlated as possible. This will introduce greater entropy into the system and will make it harder for an attacker to monitor and gather side channel information.

### 5.4.2. Pervasive Monitoring

A key additional threat consideration for group communication is pointed to by [RFC7258] which warns of the dangers of pervasive monitoring. CoAP group communication solutions which are built on top of IP multicast need to pay particular heed to these dangers. This is because IP multicast is easier to intercept (e.g., and to secretly record) compared to unicast traffic. Also, CoAP traffic is meant for the Internet of Things. This means that CoAP traffic (once future security solutions are developed as in Section 5.3.3) may be used for the control and monitoring of critical infrastructure (e.g., lights, alarms, etc.) which may be prime targets for attack.

For example, an attacker may attempt to record all the CoAP traffic going over the smart grid (i.e., networked electrical utility) of a country and try to determine critical nodes for further attacks. For example, the source node (controller) sending out the CoAP group communication messages. CoAP multicast traffic is inherently more vulnerable (compared to a unicast packet) as the same packet may be replicated over many links so there is a much higher probability of it getting captured by a pervasive monitoring system.

One useful mitigation to pervasive monitoring is to restrict the scope of the IP multicast to the minimal scope that fulfills the application need. Thus, for example, site-local IP multicast scope is always preferred over global scope IP multicast if this fulfills the application needs. This approach has the added advantage that it coincides with the guidelines for minimizing congestion control (see Section 2.8).

In the future, even if all the CoAP multicast traffic is encrypted, an attacker may still attempt to capture the traffic and perform an off-line attack. Though of course having the multicast traffic protected is always desirable as it significantly raises the cost to an attacker (e.g., to break the encryption) versus unprotected multicast traffic.

## 6. IANA Considerations

### 6.1. New 'core.gp' Resource Type

This memo registers a new resource type (rt) from the CoRE Parameters Registry called 'core.gp'.

(Note to IANA/RFC Editor: This registration follows the process described in section 7.4 of [RFC6690]).

Attribute Value: core.gp

Description: Group Configuration resource. This resource is used to query/manage the group membership of a CoAP server.

Reference: See Section 2.6.2.

### 6.2. New 'coap-group+json' Internet Media Type

This memo registers a new Internet Media Type for CoAP group configuration resource called 'application/coap-group+json'.

(Note to IANA/RFC Editor: This registration follows the guidance from [RFC6838], and (last paragraph) of Section 12.3 of [RFC7252].

Type name: application

Subtype name: coap-group+json

Required parameters: None

Optional parameters: None

Encoding considerations: 8bit UTF-8.

JSON to be represented using UTF-8 which is 8bit compatible (and most efficient for resource constrained implementations).

Security considerations:

Denial of Service attacks could be performed by constantly (re-)setting the group configuration resource of a CoAP endpoint to different values. This will cause the endpoint to register (or de-register) from the related IP multicast group. To prevent this it is recommended that a form of authorization (making use of unicast DTLS-secured CoAP) be used such that only authorized controllers are allowed by an endpoint to configure its group membership.

Interoperability considerations: None

Published specification: (This I-D when it becomes an RFC)

Applications that use this media type:

CoAP client and server implementations that wish to set/read the group configuration resource via 'application/coap-group+json' payload as described in Section 2.6.2.

Fragment identifier considerations: N/A

Additional Information:

Deprecated alias names for this type: None

Magic number(s): None

File extension(s): \*.json

Macintosh file type code(s): TEXT

Person and email address to contact for further information: Esko Dijk ("Esko.Dijk@Philips.com")

Intended usage: COMMON

Restrictions on usage: None

Author: CoRE WG

Change controller: IETF

Provisional registration? (standards tree only): N/A

## 7. Acknowledgements

Thanks to Jari Arkko, Peter Bigot, Anders Brandt, Ben Campbell, Angelo Castellani, Alissa Cooper, Spencer Dawkins, Badis Djamaa, Adrian Farrel, Stephen Farrell, Thomas Fossati, Brian Haberman, Bjoern Hoehrmann, Matthias Kovatsch, Guang Lu, Salvatore Loreto, Kerry Lynn, Andrew McGregor, Kathleen Moriarty, Pete Resnick, Dale Seed, Martin Stiemerling, Zach Shelby, Peter van der Stok, Gengyu Wei, and Juan Carlos Zuniga for their helpful comments and discussions that have helped shape this document.

Special thanks to Carsten Bormann and Barry Leiba for their extensive and thoughtful Chair and AD reviews of the document. Their reviews helped to immeasurably improve the document quality.

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002.
- [RFC3433] Bierman, A., Romascanu, D., and K. Norseth, "Entity Sensor Management Information Base", RFC 3433, December 2002.
- [RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.
- [RFC3810] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5110] Savola, P., "Overview of the Internet Multicast Routing Architecture", RFC 5110, January 2008.
- [RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, March 2010.
- [RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010.
- [RFC6092] Woodyatt, J., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, January 2011.
- [RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, March 2012.
- [RFC6636] Asaeda, H., Liu, H., and Q. Wu, "Tuning the Behavior of the Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) for Routers in Mobile and Wireless Networks", RFC 6636, May 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, May 2014.
- [RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, July 2014.

## 8.2. Informative References

- [RFC1033] Lottor, M., "Domain administrators operations guide", RFC 1033, November 1987.
- [RFC4605] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")", RFC 4605, August 2006.
- [RFC5740] Adamson, B., Bormann, C., Handley, M., and J. Macker, "NACK-Oriented Reliable Multicast (NORM) Transport Protocol", RFC 5740, November 2009.
- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-15 (work in progress), July 2014.
- [I-D.ietf-core-resource-directory]  
Shelby, Z., Bormann, C., and S. Krco, "CoRE Resource Directory", draft-ietf-core-resource-directory-01 (work in progress), December 2013.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.

[I-D.ietf-roll-trickle-mcast]

Hui, J. and R. Kelsey, "Multicast Protocol for Low power and Lossy Networks (MPL)", draft-ietf-roll-trickle-mcast-09 (work in progress), April 2014.

[I-D.keoh-dice-multicast-security]

Keoh, S., Kumar, S., Garcia-Morchon, O., Dijk, E., and A. Rahman, "DTLS-based Multicast Security in Constrained Environments", draft-keoh-dice-multicast-security-08 (work in progress), July 2014.

[I-D.droms-6man-multicast-scopes]

Droms, R., "IPv6 Multicast Address Scopes", draft-droms-6man-multicast-scopes-02 (work in progress), July 2013.

[I-D.mglt-dice-ipsec-for-application-payload]

Migault, D. and C. Bormann, "IPsec/ESP for Application Payload", draft-mglt-dice-ipsec-for-application-payload-00 (work in progress), July 2014.

#### Appendix A. Multicast Listener Discovery (MLD)

In order to extend the scope of IP multicast beyond link-local scope, an IP multicast routing or forwarding protocol has to be active in routers on an LLN. To achieve efficient IP multicast routing (i.e., avoid always flooding IP multicast packets), routers have to learn which hosts need to receive packets addressed to specific IP multicast destinations.

The Multicast Listener Discovery (MLD) protocol [RFC3810] (or its IPv4 equivalent IGMP [RFC3376]) is today the method of choice used by an (IP multicast enabled) router to discover the presence of IP multicast listeners on directly attached links, and to discover which IP multicast addresses are of interest to those listening nodes. MLD was specifically designed to cope with fairly dynamic situations in which IP multicast listeners may join and leave at any time.

[RFC6636] discusses optimal tuning of the parameters of MLD/IGMP for routers for mobile and wireless networks. These guidelines may be useful when implementing MLD in LLNs.

#### Appendix B. Change Log

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-24 to ietf-25:



- o Updated with the remaining agreed minor comments from Ben Campbell's GEN-ART review. Specifically, addressed the two comments on section 2.6.2.1 (which was section 2.7.2.1 in rev-21) as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05604.html>".
- o Updated with the clarification comment from Badis Djamaa in Section 2.3 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05612.html>".
- o Minor editorial updates.

Changes from ietf-23 to ietf-24:

- o Clarified in section 2.6.1.2 (Configuring Members) that ABNF rules from Section 3.2.2 of [RFC 3986] should be used for the IP address parsing.
- o Minor editorial updates.

Changes from ietf-22 to ietf-23:

- o Updated requirements language (RFC 2119) to follow Barry Leiba's suggestions #1, #2b, and #2.1 as per "<http://www.ietf.org/mail-archive/web/core/current/msg05593.html>".
- o Clarified that [RFC 7320] implies that also other specifications cannot pre-define URI structure.
- o Added MUST to Token re-use time as it is additional specification to CoAP [RFC 7252].
- o Clarified use of multicast POSTing in Section 2.4, in response to Jari Arkko's COMMENTS in "<http://www.ietf.org/mail-archive/web/core/current/msg05572.html>".
- o Added to Section 5.1 (Security Configuration) the possibility to use application-layer (data object) security, which enables to use CoAP group communication also for critical applications, pointed out by Jari Arkko's COMMENTS in "<http://www.ietf.org/mail-archive/web/core/current/msg05572.html>".
- o Fixed subtle error in hex string "c00l" to "c001".
- o Clarified in Section 2.11 (Exceptions) that CoAP Observe feature does not support group communication as per Jari's Arkko's comment in "<http://www.ietf.org/mail-archive/web/core/current/msg05592.html>".

- o Moved section 2.6 (Member Discovery) into a new subsection as part of 2.7.1 (Membership Configuration - Background).
- o Minor editorial updates.

Changes from ietf-21 to ietf-22:

- o Updated with comments from IESG review as follows:
  1. Changed Status from Informational to Experimental.
  2. Addressed Brian Haberman's DISCUSS (to put in reference to ASM) in section 1.2 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05547.html>".
  3. Addressed Brian Haberman's DISCUSS (to put in reference to multicast forwarding proxies) in section 2.1 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05547.html>".
  4. Addressed Brian Haberman's DISCUSS (to put in reference to getting port numbers from URIs) in section 2.3 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05563.html>".
  5. Addressed Brian Haberman's DISCUSS (to put in reference to IGMP/MLD API) in section 2.7.2.1, 2.7.2.2, 2.7.2.6 and 2.7.2.7 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05547.html>".
  6. Addressed Brian Haberman's COMMENT (to put in reference to reliable multicast RFC) in section 1.3 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05545.html>".
  7. Addressed Kathleen Moriarty's DISCUSS (to broaden to cover general monitoring) in section 5.4 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05566.html>".
  8. Addressed Martin Stiemerling's DISCUSS (to clearly indicate that the draft introduces new CoAP protocol functionality) in the Abstract and section 1.2 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05542.html>".
  9. Addressed Martin Stiemerling's DISCUSS (to clarify selected requirements language) in section 2.7.2 as called out in

"<http://www.ietf.org/mail-archive/web/core/current/msg05542.html>". (Note that the other sections are not impacted as they truly are new requirements and not repetition of the CoAP RFC 7252)

10. Addressed Spencer Dawkins' COMMENT as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05557.html>".
  11. Addressed Alissa Cooper's COMMENT as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05567.html>".
  12. Addressed selected Stephen Farrell's COMMENTS as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05576.html>".
  13. Addressed selected Pete Resnick's COMMENTS as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05568.html>".
  14. Addressed selected Adrian Farrel's COMMENTS as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05565.html>".
  15. Addressed selected Jari Arkko's COMMENTS as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05572.html>".
- o Updated with comments from GEN-ART review as follows:
1. Addressed major issue #1 from Ben Campbell's GEN-ART review (about introducing new functionality beyond CoAP RFC 7252) by changing the status of document to Experimental, and updating Abstract and section 2.1 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05551.html>".
  2. Addressed major issue #2 from Ben Campbell's GEN-ART review (about giving a stronger recommendation not to use CoAP group communication in many scenarios until stronger security solutions are available) in section 5.1 and section 5.4 as called out in "<http://www.ietf.org/mail-archive/web/core/current/msg05551.html>".
  3. Addressed selected minor issues and nits from Ben Campbell's GEN-ART review comments from "<http://www.ietf.org/mail-archive/web/core/current/msg05535.html>".

- o Various minor editorial updates.

Changes from ietf-20 to ietf-21:

- o Updated with comments from AD review by Barry Leiba. The details of the updates can be seen by looking at the WG mailing list from July 26-31, 2014.
- o Various minor editorial updates.

Changes from ietf-19 to ietf-20:

- o Replaced obsolete reference [RFC 2616] by [RFC 7230].
- o Replaced outdated reference draft-ietf-appsawg-uri-get-off-my-lawn by [RFC 7320] and moved to Normative reference.
- o Replaced outdated reference draft-ietf-core-coap by [RFC 7252].
- o Moved [RFC 1033] to Informative reference.
- o Updated to latest revision numbers for informative draft references by regenerating file through xml2rfc tool.
- o Re-ran IETF spell check tool and corrected some minor spelling errors.
- o Various minor editorial updates.

Changes from ietf-18 to ietf-19:

- o Added guideline on Token value re-use in section 2.5.
- o Updated section 5.1 (Security Configuration) and 5.3.3 (Future Security Evolution) to point to latest security developments happening in DICE WG for support of group security.
- o Added Pervasive Monitoring considerations in section 5.4.
- o Various editorial updates for improved readability.

Changes from ietf-17 to ietf-18:

- o Extensive editorial updates based on WGLC comments by Thomas Fossati and Gengyu Wei.
- o Addressed ticket #361: Added text for single membership PUT section 2.7.2.7 (Updating a single group membership (PUT)).

- o Addressed ticket #360: Added text for server duties upon all-at-once PUT section 2.7.2.6 (Creating/updating all group memberships at once (PUT)).
- o Addressed ticket #359: Fixed requirements text for Section 2.7.2.2 (Creating a new multicast group membership (POST)).
- o Addressed ticket #358: Fixed requirements text for Section 2.7.2.1 (CoAP-Group Resource Type and Media Type).
- o Addressed ticket #357: Added that "IPv6 addresses of other scopes MAY be enabled" in section 2.2 (Group Definition and Naming).
- o Various editorial updates for improved readability.

Changes from ietf-16 to ietf-17:

- o Added guidelines on joining of IPv6/IPv4 "All CoAP Nodes" multicast addresses (#356).
- o Added MUST support default port in case multicast discovery is available.
- o In section 2.1 (IP Multicast Background), clarified that IP multicast is not guaranteed and referenced a definition of Reliable Group Communication (#355).
- o Added section 2.5 (Messages and Responses) to clarify how responses are identified and how Token/MID are used in multicast CoAP.
- o In section 2.6.2 (RESTful Interface for Configuring Group Memberships), clarified that group management interface is an optional approach for dynamic commissioning and that other approaches can also be used if desired.
- o Updated section 2.6.2 (RESTful Interface for Configuring Group Memberships) to allow deletion of individual group memberships (#354).
- o Various editorial updates based on comments by Peter van der Stok. Removed reference to expired draft-vanderstok-core-dna at request of its author.
- o Various editorial updates for improved readability.

Changes from ietf-15 to ietf-16:

- o In section 2.6.2, changed DELETE in group management interface to a PUT with empty JSON array to clear the list (#345).
- o In section 2.6.2, aligned the syntax for IP addresses to follow RFC 3986 URI syntax, which is also used by coap-18. This allows re-use of the parsing code for CoAP URIs for this purpose (#342).
- o Addressed some more editorial comments provided by Carsten Bormann in preparation for WGLC.
- o Various editorial updates for improved readability.

Changes from ietf-14 to ietf-15:

- o In section 2.2, provided guidance on how implementers should parse URIs for group communication (#339).
- o In section 2.6.2.1, specified that for group membership configuration interface the "ip" (i.e., "a" parameter) key/value is not required when it is unknown (#338).
- o In section 2.6.2.1, specified that for group membership configuration interface the port configuration be defaulted to standard CoAP port 5683, and if not default then should follow standard notation (#340).
- o In section 2.6.2.1, specified that notation of IP address in group membership configuration interface should follow standard notation (#342).
- o In section 6.2, "coap-group+json" Media Type encoding simplified to just support UTF-8 (and not UTF-16 and UTF-32) (#344).
- o Various editorial updates for improved readability.

Changes from ietf-13 to ietf-14:

- o Update to address final editorial comments from the Chair's review (by Carsten Bormann) of the draft. This included restructuring of Section 2.6 (Configuring Group Memberships) and Section 4 (Deployment Guidelines) to make it easier to read. Also various other editorial changes.
- o Changed "ip" field to "a" in Section 2.6 (#337)

Changes from ietf-12 to ietf-13:

- o Extensive editorial updates due to comments from the Chair's review (by Carsten Bormann) of the draft. The best way to see the changes will be to do a -Diff with Rev. 12.
- o The technical comments from the Chair's review will be addressed in a future revision after tickets are generated and the solutions are agreed to on the WG E-mail list.

Changes from ietf-11 to ietf-12:

- o Removed reference to "CoAP Ping" in Section 3.5 (Group Member Discovery) and replaced it with the more efficient support of discovery of groups and group members via the CORE RD as suggested by Zach Shelby.
- o Various editorial updates for improved readability.

Changes from ietf-10 to ietf-11:

- o Added text to section 3.8 (Congestion Control) to clarify that a "CoAP client sending a multicast CoAP request to /.well-known/core SHOULD support core-block" (#332).
- o Various editorial updates for improved readability.

Changes from ietf-09 to ietf-10:

- o Various editorial updates including:
- o Added a fourth option in section 3.3 on ways to obtain the URI path for a group request.
- o Clarified use of content format in GET/PUT requests for Configuring Group Membership in Endpoints (in section 3.6).
- o Changed reference "draft-shelby-core-resource-directory" to "draft-ietf-core-resource-directory".
- o Clarified (in section 3.7) that ACKs are never used for a multicast request (from #296).
- o Clarified (in section 5.2/5.2.3) that MPL does not support group membership advertisement.
- o Adding introductory paragraph to Scope (section 2.2).
- o Wrote out fully the URIs in table section 3.2.

- o Reworded security text in section 7.2 (New Internet Media Type) to make it consistent with section 3.6 (Configuring Group Membership).
- o Fixed formatting of hyperlinks in sections 6.3 and 7.2.

Changes from ietf-08 to ietf-09:

- o Cleaned up requirements language in general. Also, requirements language are now only used in section 3 (Protocol Considerations) and section 6 (Security Considerations). Requirements language has been removed from other sections to keep them to a minimum (#271).
- o Addressed final comment from Peter van der Stok to define what "IP stack" meant (#296). Following the lead of CoAP-17, we now refer instead to "APIs such as IPV6\_RECVPKTINFO [RFC 3542]".
- o Changed text in section 3.4 (Group Methods) to allow multicast POST under specific conditions and highlighting the risks with using it (#328).
- o Various editorial updates for improved readability.

Changes from ietf-07 to ietf-08:

- o Updated text in section 3.6 (Configuring Group Membership in Endpoints) to make it more explicit that the Internet Media Type is used in the processing rules (#299).
- o Addressed various comments from Peter van der Stok (#296).
- o Various editorial updates for improved readability including defining all acronyms.

Changes from ietf-06 to ietf-07:

- o Added an IANA request (in section 7.2) for a dedicated content-format (Internet Media type) for the group management JSON format called 'application/coap-group+json' (#299).
- o Clarified semantics (in section 3.6) of group management JSON format (#300).
- o Added details of IANA request (in section 7.1) for a new CORE Resource Type called 'core.gp'.



- o Clarified that DELETE method (in section 3.6) is also a valid group management operation.
- o Various editorial updates for improved readability.

Changes from ietf-05 to ietf-06:

- o Added a new section on commissioning flow when using discovery services when end devices discover in which multicast group they are allocated (#295).
- o Added a new section on CoAP Proxy Operation (section 3.9) that outlines the potential issues and limitations of doing CoAP multicast requests via a CoAP Proxy (#274).
- o Added use case of multicasting controller on the backbone (#279).
- o Use cases were updated to show only a single CoAP RD (to replace the previous multiple RDs with one in each subnet). This is a more efficient deployment and also avoids RD specific issues such as synchronization of RD information between serves (#280).
- o Added text to section 3.6 (Configuring Group Membership in Endpoints) that clarified that any (unicast) operation to change an endpoint's group membership must use DTLS-secured CoAP.
- o Clarified relationship of this document to draft-ietf-core-coap in section 2.2 (Scope).
- o Removed IPSec related requirement, as IPSec is not part of draft-ietf-core-coap anymore.
- o Editorial reordering of subsections in section 3 to have a better flow of topics. Also renamed some of the (sub)sections to better reflect their content. Finally, moved the URI Configuration text to the same section as the Port Configuration section as it was a more natural grouping (now in section 3.3) .
- o Editorial rewording of section 3.7 (Multicast Request Acceptance and Response Suppression) to make the logic easier to comprehend (parse).
- o Various editorial updates for improved readability.

Changes from ietf-04 to ietf-05:

- o Added a new section 3.9 (Exceptions) that highlights that IP multicast (and hence group communication) is not always available (#187).
- o Updated text on the use of [RFC2119] language (#271) in Section 1.
- o Included guidelines on when (not) to use CoAP responses to multicast requests and when (not) to accept multicast requests (#273).
- o Added guideline on use of core-block for minimizing response size (#275).
- o Restructured section 6 (Security Considerations) to more fully describe threats and threat mitigation (#277).
- o Clearly indicated that DNS resolution and reverse DNS lookup are optional.
- o Removed confusing text about a single group having multiple IP addresses. If multiple IP addresses are required then multiple groups (with the same members) should be created.
- o Removed repetitive text about the fact that group communication is not guaranteed.
- o Merged previous section 5.2 (Multicast Routing) into 3.1 (IP Multicast Routing Background) and added link to section 5.2 (Advertising Membership of Multicast Groups).
- o Clarified text in section 3.8 (Congestion Control) regarding precedence of use of IP multicast domains (i.e., first try to use link-local scope, then site-local scope, and only use global IP multicast as a last resort).
- o Extended group resource manipulation guidelines with use of pre-configured ports/paths for the multicast group.
- o Consolidated all text relating to ports in a new section 3.3 (Port Configuration).
- o Clarified that all methods (GET/PUT/POST) for configuring group membership in endpoints should be unicast (and not multicast) in section 3.7 (Configuring Group Membership In Endpoints).
- o Various editorial updates for improved readability, including editorial comments by Peter van der Stok to WG list of December 18th, 2012.

Changes from ietf-03 to ietf-04:

- o Removed section 2.3 (Potential Solutions for Group Communication) as it is purely background information and moved section to draft-dijk-core-groupcomm-misc (#266).
- o Added reference to draft-keoh-tls-multicast-security to section 6 (Security Considerations).
- o Removed Appendix B (CoAP-Observe Alternative to Group Communications) as it is as an alternative to IP Multicast that the WG has not adopted and moved section to draft-dijk-core-groupcomm-misc (#267).
- o Deleted section 8 (Conclusions) as it is redundant (#268).
- o Simplified light switch use case (#269) by splitting into basic operations and additional functions (#269).
- o Moved section 3.7 (CoAP Multicast and HTTP Unicast Interworking) to draft-dijk-core-groupcomm-misc (#270).
- o Moved section 3.3.1 (DNS-SD) and 3.3.2 (CoRE Resource Directory) to draft-dijk-core-groupcomm-misc as these sections essentially just repeated text from other drafts regarding DNS based features. Clarified remaining text in this draft relating to DNS based features to clearly indicate that these features are optional (#272).
- o Focus section 3.5 (Configuring Group Membership) on a single proposed solution.
- o Scope of section 5.3 (Use of MLD) widened to multicast destination advertisement methods in general.
- o Rewrote section 2.2 (Scope) for improved readability.
- o Moved use cases that are not addressed to draft-dijk-core-groupcomm-misc.
- o Various editorial updates for improved readability.

Changes from ietf-02 to ietf-03:

- o Clarified that a group resource manipulation may return back a mixture of successful and unsuccessful responses (section 3.4 and Figure 6) (#251).

- o Clarified that security option for group communication must be NoSec mode (section 6) (#250).
- o Added mechanism for group membership configuration (#249).
- o Removed IANA request for multicast addresses (section 7) and replaced with a note indicating that the request is being made in draft-ietf-core-coap (#248).
- o Made the definition of 'group' more specific to group of CoAP endpoints and included text on UDP port selection (#186).
- o Added explanatory text in section 3.4 regarding why not to use group communication for non-idempotent messages (i.e., CoAP POST) (#186).
- o Changed link-local RD discovery to site-local in RD discovery use case to make it more realistic.
- o Fixed lighting control use case CoAP proxying; now returns individual CoAP responses as in coap-12.
- o Replaced link format I-D with RFC6690 reference.
- o Various editorial updates for improved readability

Changes from ietf-01 to ietf-02:

- o Rewrote congestion control section based on latest CoAP text including Leisure concept (#188)
- o Updated the CoAP/HTTP interworking section and example use case with more details and use of MLD for multicast group joining
- o Key use cases added (#185)
- o References to draft-vanderstok-core-dna and draft-castellani-core-advanced-http-mapping added
- o Moved background sections on "MLD" and "CoAP-Observe" to Appendices
- o Removed requirements section (and moved it to draft-dijk-core-groupcomm-misc)
- o Added details for IANA request for group communication multicast addresses

- o Clarified text to distinguish between "link local" and general multicast cases
- o Moved lengthy background section 5 to draft-dijk-core-groupcomm-misc and replaced with a summary
- o Various editorial updates for improved readability
- o Change log added

Changes from ietf-00 to ietf-01:

- o Moved CoAP-observe solution section to section 2
- o Editorial changes
- o Moved security requirements into requirements section
- o Changed multicast POST to PUT in example use case
- o Added CoAP responses in example use case

Changes from rahman-07 to ietf-00:

- o Editorial changes
- o Use cases section added
- o CoRE Resource Directory section added
- o Removed section 3.3.5. IP Multicast Transmission Methods
- o Removed section 3.4 Overlay Multicast
- o Removed section 3.5 CoAP Application Layer Group Management
- o Clarified section 4.3.1.3 RPL Routers with Non-RPL Hosts case
- o References added and some normative/informative status changes

#### Authors' Addresses

Akbar Rahman (editor)  
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

Esko Dijk (editor)  
Philips Research

Email: [esko.dijk@philips.com](mailto:esko.dijk@philips.com)

CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: December 3, 2012

Z. Shelby  
Sensinode  
June 1, 2012

CoRE Link Format  
draft-ietf-core-link-format-14

Abstract

This specification defines Web Linking using a link format for use by constrained web servers to describe hosted resources, their attributes and other relationships between links. Based on the HTTP Link Header field defined in RFC5988, the CoRE Link Format is carried as a payload and is assigned an Internet media type. A well-known URI is defined as a default entry-point for requesting the links hosted by a server.

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 December 3, 2012.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .   | 3  |
| 1.1. Web Linking in CoRE . . . . .  | 3  |
| 1.2. Use Cases . . . . .  | 4  |
| 1.2.1. Discovery . . . . .  | 4  |
| 1.2.2. Resource Collections . . . . .   | 5  |
| 1.2.3. Resource Directory . . . . .   | 5  |
| 1.3. Terminology . . . . .  | 5  |
| 2. Link Format . . . . .  | 6  |
| 2.1. Target and context URIs . . . . .  | 9  |
| 2.2. Link relations . . . . .   | 9  |
| 2.3. Use of anchors . . . . .   | 9  |
| 3. CoRE link attributes . . . . .   | 9  |
| 3.1. Resource type 'rt' attribute . . . . .                                   | 10 |
| 3.2. Interface description 'if' attribute . . . . .                           | 10 |
| 3.3. Maximum size estimate 'sz' attribute . . . . .                           | 11 |
| 4. Well-known Interface . . . . .   | 11 |
| 4.1. Query Filtering . . . . .  | 12 |
| 5. Examples . . . . .   | 13 |
| 6. Security Considerations . . . . .  | 15 |
| 7. IANA Considerations . . . . .  | 16 |
| 7.1. Well-known 'core' URI . . . . .  | 16 |
| 7.2. New 'hosts' relation type . . . . .                                      | 16 |
| 7.3. New link-format Internet media type . . . . .                            | 17 |
| 7.4. Constrained RESTful Environments (CORE) Parameters<br>Registry . . . . . | 18 |
| 8. Acknowledgments . . . . .  | 19 |
| 9. Changelog . . . . .  | 20 |
| 10. References . . . . .  | 24 |
| 10.1. Normative References . . . . .  | 24 |
| 10.2. Informative References . . . . .  | 24 |
| Author's Address . . . . .  | 25 |



## 1. Introduction

The Constrained RESTful Environments (CoRE) realizes the Representational State Transfer (REST) architecture [REST] in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited memory) and networks (e.g. 6LoWPAN [RFC4919]). CoRE is aimed at Machine-to-Machine (M2M) applications such as smart energy and building automation.

The discovery of resources hosted by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP [RFC2616] Web Server is typically called Web Discovery and the description of relations between resources is called Web Linking [RFC5988]. In the present specification we refer to the discovery of resources hosted by a constrained web server, their attributes and other resource relations as CoRE Resource Discovery.

The main function of such a discovery mechanism is to provide Universal Resource Identifiers (URIs, called links) for the resources hosted by the server, complemented by attributes about those resources and possible further link relations. In CoRE this collection of links is carried as a resource of its own (as opposed to HTTP headers delivered with a specific resource). This document specifies a link format for use in CoRE Resource Discovery by extending the HTTP Link Header format [RFC5988] to describe these link descriptions. The CoRE Link Format is carried as a payload and is assigned an Internet media type. A well-known relative URI `"/.well-known/core"` is defined as a default entry-point for requesting the list of links about resources hosted by a server, and thus performing CoRE Resource Discovery. This specification is applicable for use with Constrained Application Protocol (CoAP) [I-D.ietf-core-coap], HTTP or any other suitable web transfer protocol. The link format can also be saved in file format.

### 1.1. Web Linking in CoRE

Technically the CoRE Link Format is a serialization of a typed link as specified in [RFC5988], used to describe relationships between resources, so-called "Web Linking". In this specification Web Linking is extended with specific constrained M2M attributes, links are carried as a message payload rather than in an HTTP Link Header field, and a default interface is defined to discover resources hosted by a server. This specification also defines a new relation type "hosts" (from the verb "to host"), which indicates that the resource is hosted by the server from which the link document was requested.

In HTTP, the Link Header can be used to carry link information about a resource along with an HTTP response. This works well for the typical use case for a web server and browser, where further information about a particular resource is useful after accessing it. In CoRE the main use case for Web Linking is the discovery of which resources a server hosts in the first place. Although some resources may have further links associated with them, this is expected to be an exception. For that reason the CoRE Link Format serialization is carried as a resource representation of a well-known URI. The CoRE Link Format does re-use the format of the HTTP Link Header serialization defined in [RFC5988].

## 1.2. Use Cases

Typical use cases for Web Linking on today's web include e.g. describing the author of a web page or describing relations between web pages (next chapter, previous chapter etc.). Web Linking can also be applied to M2M applications, where typed links are used to assist a machine client in finding and understanding how to use resources on a server. In this section a few use cases are described for how the CoRE Link Format could be used in M2M applications. For further technical examples see Section 5. As there are a large range of M2M applications, these use cases are purposely generic. This specification assumes that different deployments or application domains will define the appropriate REST Interface Descriptions along with Resource Types to make discovery meaningful.

### 1.2.1. Discovery

In M2M applications, for example home or building automation, there is a need for local clients and servers to find and interact with each other without human intervention. The CoRE Link Format can be used by servers in such environments to enable Resource Discovery of the resources hosted by the server.

Resource Discovery can be performed either unicast or multicast. When a server's IP address is already known, either a priori or resolved via the Domain Name System (DNS) [RFC1034][RFC1035], unicast discovery is performed in order to locate the entry point to the resource of interest. In this specification, this is performed using a GET to `"/.well-known/core"` on the server, which returns a payload in the CoRE Link Format. A client would then match the appropriate Resource Type, Interface Description and possible Media type [RFC2045] for its application. These attributes may also be included in the query string in order to filter the number of links returned in a response.

Multicast resource discovery is useful when a client needs to locate

a resource within a limited scope, and that scope supports IP multicast. A GET request to the appropriate multicast address is made for `"/.well-known/core"`. In order to limit the number and size of responses, a query string is recommended with the known attributes. Typically a resource would be discovered based on its Resource Type and/or Interface Description, along with possible application specific attributes.

#### 1.2.2. Resource Collections

RESTful designs of M2M interfaces often make use of collections of resources. For example an index of temperature sensors on a data collection node or a list of alarms on a home security controller. The CoRE Link Format can be used to make it possible to find the entry point to a collection and traverse its members. The entry point of a collection would always be included in `"/.well-known/core"` to enable its discovery. The members of the collection can be defined either through the Interface Description of the resource along with a parameter resource for the size of the collection, or by using the link format to describe each resource in the collection. These links could be located under `"/.well-known/core"` or hosted for example in the root resource of the collection.

#### 1.2.3. Resource Directory

In many deployment scenarios, for example constrained networks with sleeping servers, or large M2M deployments with bandwidth limited access networks, it makes sense to deploy resource directory entities which store links to resources stored on other servers. Think of this as a limited search engine for constrained M2M resources.

The CoRE Link Format can be used by a server to register resources with a resource directory, or to allow a resource directory to poll for resources. Resource registration can be achieved by having each server POST their resources to `"/.well-known/core"` on the resource directory. This in turn adds links to the resource directory under an appropriate resource. These links can then be discovered by any client by making a request to a resource directory lookup interface.

#### 1.3. Terminology

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

This specification makes use of the Augmented Backus-Naur Form (ABNF) [RFC5234] notation, including the core rules defined in Appendix A of that document.

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6454]. In addition, this specification makes use of the following terminology:

**Web Linking**

A framework for indicating the relationships between web resources.

**Link**

Also called "typed links" in RFC5988. A link is a typed connection between two resources identified by URIs. Made up of a context URI, a link relation type, a target URI, and optional target attributes.

**Link Format**

A particular serialization of typed links.

**CoRE Link Format**

A particular serialization of typed links based on the HTTP Link Header field serialization defined in Section 5 of RFC5988, but carried as a resource representation with a media type.

**Attribute**

Properly called "Target Attribute" in RFC5988. A key/value pair that describes the link or its target.

**CoRE Resource Discovery**

When a client discovers the list of resources hosted by a server, their attributes and other link relations by accessing `"/.well-known/core"`.

## 2. Link Format

The CoRE Link Format extends the HTTP Link Header field specified in [RFC5988]. The format does not require special XML or binary parsing, is fairly compact, and is extensible - all important characteristics for CoRE. It should be noted that this link format is just one serialization of typed links defined in [RFC5988], others include HTML link, Atom feed links [RFC4287] or HTTP Link Header fields. It is expected that resources discovered in the CoRE Link Format may also be made available in alternative formats on the greater Internet. The CoRE Link Format is only expected to be supported in constrained networks and M2M systems.

Section 5 of [RFC5988] did not require an Internet media type for the defined link format, as it was defined to be carried in an HTTP header. This specification thus defines the Internet media type

"application/link-format" for the CoRE Link Format (see Section 7.3). Whereas the HTTP Link Header field depends on [RFC2616] for its encoding, the CoRE Link Format is encoded as UTF-8 [RFC3629]. A decoder of the format is not expected to (but not prohibited from) validate UTF-8 encoding and doesn't need to perform any UTF-8 normalization. UTF-8 data can be compared bit-wise, which allows values to contain UTF-8 data without any added complexity for constrained nodes.

The CoRE link format is equivalent to the [RFC5988] link format, however the ABNF in the present specification is repeated with improvements to be compliant with [RFC5234] and includes new link parameters. The link parameter "href" is reserved for use as a query parameter for filtering in this specification (see Section 4.1), and MUST NOT be defined as a link parameter. As in [RFC5988], multiple link descriptions are separated by commas. Note that commas can also occur in quoted strings and URIs but do not end a description. In order to convert an HTTP Link Header field to this link format, first the "Link:" HTTP header is removed, any LWS is removed, the header value is converted to UTF-8 and any percent-encodings decoded.

```

Link           = link-value-list
link-value-list = [ link-value *[ "," link-value ] ]
link-value     = "<" URI-Reference ">" *( ";" link-param )
link-param     = ( ( "rel" "=" relation-types )
/ ( "anchor" "=" DQUOTE URI-Reference DQUOTE )
/ ( "rev" "=" relation-types )
/ ( "hreflang" "=" Language-Tag )
/ ( "media" "=" ( MediaDesc
/ ( DQUOTE MediaDesc DQUOTE ) ) )
/ ( "title" "=" quoted-string )
/ ( "title*" "=" ext-value )
/ ( "type" "=" ( media-type / quoted-mt ) )
/ ( "rt" "=" relation-types )
/ ( "if" "=" relation-types )
/ ( "sz" "=" cardinal )
/ ( link-extension ) )
link-extension = ( parmname [ "=" ( ptoken / quoted-string ) ] )
/ ( ext-name-star "=" ext-value )
ext-name-star  = parmname "*" ; reserved for RFC2231-profiled
/ extensions. Whitespace NOT
/ allowed in between.

ptoken         = 1*ptokenchar
ptokenchar     = "!" / "#" / "$" / "%" / "&" / "'" / "("
/ ")" / "*" / "+" / "-" / "." / "/" / DIGIT
/ ":" / "<" / "=" / ">" / "?" / "@" / ALPHA
/ "[" / "]" / "^" / "_" / "`" / "{" / "|"
/ "}" / "~"

media-type     = type-name "/" subtype-name
quoted-mt      = DQUOTE media-type DQUOTE
relation-types = relation-type
/ DQUOTE relation-type *( 1*SP relation-type ) DQUOTE
relation-type  = reg-rel-type / ext-rel-type
reg-rel-type   = LOALPHA *( LOALPHA / DIGIT / "." / "-" )
ext-rel-type   = URI
cardinal       = "0" / ( %x31-39 *DIGIT )
LOALPHA        = %x61-7A ; a-z
quoted-string  = <defined in RFC2616>
URI            = <defined in RFC3986>
URI-Reference  = <defined in RFC3986>
type-name      = <defined in RFC4288>
subtype-name   = <defined in RFC4288>
MediaDesc      = <defined in W3C.REC-html401-19991224>
Language-Tag   = <defined in RFC5646>
ext-value      = <defined in RFC5987>
parmname       = <defined in RFC5987>

```

### 2.1. Target and context URIs

Each link conveys one target URI as a URI-reference inside angle brackets ("`<>`"). The context URI of a link (also called base URI in [RFC3986]) is determined by the following rules in this specification:

- (a) The context URI is set to the anchor parameter, when specified, or
- (b) Origin of the target URI, when specified
- (c) Origin of the link format resource's base URI.

### 2.2. Link relations

Since links in the CoRE Link Format are typically used to describe resources hosted by a server, and thus in the absence of the relation parameter the new relation type "hosts" is assumed (see Section 7.2). The "hosts" relation type (from the verb "to host") indicates that the target URI is a resource hosted by the server (i.e. server hosts resource) indicated by the context URI. The target URI MUST be a relative URI of the context URI for this relation type.

To express other relations, links can make use of any registered relation by including the relation parameter. The context of a relation can be defined using the anchor parameter. In this way, relations between resources hosted on a server, or between hosted resources and external resources can be expressed.

### 2.3. Use of anchors

As per Section 5.2 of [RFC5988] a link description MAY include an "anchor" attribute, in which case the context is the URI included in that attribute. This is used to describe a relationship between two resources. A consuming implementation can however choose to ignore such links. It is not expected that all implementations will be able to derive useful information from explicitly anchored links.

## 3. CoRE link attributes

The following CoRE specific target attributes are defined in addition to those already defined in [RFC5988]. These attributes describe information useful in accessing the target link of the relation, and in some cases can use the syntactical form of a URI. Such a URI MAY be dereferenced (for instance to obtain a description of the link relation), but that this is not part of the protocol and MUST NOT be

done automatically on link evaluation. When attributes values are compared, they MUST be compared as strings.

### 3.1. Resource type 'rt' attribute

The resource type "rt" attribute is an opaque string used to assign an application specific semantic type to a resource. One can think of this as a noun describing the resource. In the case of a temperature resource this could be e.g. an application-specific semantic type like "outdoor-temperature" or a URI referencing a specific concept in an ontology like "http://sweet.jpl.nasa.gov/2.0/phys.owl#Temperature". Multiple resource types MAY be included in the value of this parameter, each separated by a space, similar to the relation attribute. The registry for Resource Type values is defined in Section 7.4.

The resource type attribute is not meant to be used to assign a human readable name to a resource. The "title" attribute defined in [RFC5988] is meant for that purpose. The resource type attribute MUST NOT appear more than once in a link.

### 3.2. Interface description 'if' attribute

The Interface Description "if" attribute is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource. One can think of this as describing verbs usable on a resource. The Interface Description attribute is meant to describe the generic REST interface to interact with a resource or a set of resources. It is expected that an Interface Description will be re-used by different resource types. For example the resource types "outdoor-temperature", "dew-point" and "rel-humidity" could all be accessible using the interface description "http://www.example.org/myapp.wadl#sensor". Multiple interface descriptions MAY be included in the value of this parameter, each separated by a space, similar to the relation attribute. The registry for Interface Description values is defined in Section 7.4.

The Interface Description could be for example the URI of a Web Application Description Language (WADL) [WADL] definition of the target resource "http://www.example.org/myapp.wadl#sensor", a URN indicating the type of interface to the resource "urn:myapp:sensor", or an application-specific name "Sensor". The Interface Description attribute MUST NOT appear more than once in a link.



### 3.3. Maximum size estimate 'sz' attribute

The maximum size estimate attribute "sz" gives an indication of the maximum size of the resource representation returned by performing a GET on the target URI. For links to CoAP resources this attribute is not expected to be included for small resources that can comfortably be carried in a single Maximum Transmission Unit (MTU), but SHOULD be included for resources larger than that. The maximum size estimate attribute MUST NOT appear more than once in a link.

Note that there is no defined upper limit to the value of the sz attributes. Implementations MUST be prepared to accept large values. One implementation strategy is to convert any value larger than a reasonable size limit for this implementation to a special value "Big", which in further processing would indicate that a size value was given that was so big that it cannot be processed by this implementation.

## 4. Well-known Interface

Resource discovery in CoRE is accomplished through the use of a well-known resource URI which returns a list of links about resources hosted by that server and other link relations. Well-known resources have a path component that begins with "/.well-known/" as specified in [RFC5785]. This specification defines a new well-known resource for CoRE Resource Discovery "/.well-known/core".

A server implementing this specification MUST support this resource on the default port appropriate for the protocol for the purpose of resource discovery. It is however up to the application which links are included and how they are organized. The resource "/.well-known/core" is meant to be used to return links to the entry points of resource interfaces on a server. More sophisticated link organization can be achieved by including links to CoRE Link Format resources located elsewhere on the server, for example to achieve an index. In the absence of any links, a zero-length payload is returned. The resource representation of this resource MUST be the CoRE Link Format described in Section 2.

The CoRE resource discovery interface supports the following interactions:

- o Performing a GET on "/.well-known/core" to the default port returns a set of links available from the server (if any) in the CoRE Link Format. These links might describe resources hosted on that server, on other servers, or express other kinds of link relations as described in Section 2.

- o Filtering may be performed on any of the link format attributes using a query string as specified in Section 4.1. For example [GET /.well-known/core?rt=temperature-c] would request resources with the resource type TemperatureC. A server is not however required to support filtering.
- o More capable servers such as proxies could support a resource directory by requesting the resource descriptions of other endpoints or allowing servers to POST requests to "/.well-known/core". The details of such resource directory functionality is however out of scope for this specification, and is expected to be specified separately.

#### 4.1. Query Filtering

A server implementing this specification MAY recognize the query part of a resource discovery URI as a filter on the resources to be returned. The path and query components together should conform to the following level-4 URI Template [RFC6570]

```
/.well-known/core{?search*}
```

where the variable "search" is a 1-element list that has a single name/value pair, where

- o name is either "href", a link-param name defined in this specification, or any other link-extension name, and
- o value is either a Complete Value String that does not end in a "\*" (%2A), or a Prefix Value String followed by a "\*" (%2A).

The search name "href" refers to the URI-reference between the "<" and ">" characters of a link. Both Value Strings match a target attribute only if it exists. Value Strings are percent-decoded ([RFC3986] section 2.1) before matching; similarly, any target attributes notated as quoted-string are interpreted as defined in section 2.2 of [RFC2616]. After these steps, a Complete Value String matches a target attribute if it is bitwise identical. A Prefix Value String matches a target attribute if it is a bitwise prefix of the target attribute (where any string is a prefix of itself). Empty prefix value strings are allowed, by the definition above they match any target attribute that does exist. Note that relation-type target attributes can contain multiple values, and each value MUST be treated as a separate target attribute when matching.

It is not expected that very constrained nodes support filtering.

Implementations not supporting filtering MUST simply ignore the query string and return the whole resource for unicast requests.

When using a transfer protocol like the Constrained Application Protocol (CoAP) that supports multicast requests, special care needs to be taken. A multicast request with a query string SHOULD NOT be responded to if filtering is not supported or if the filter does not match (to avoid a needless response storm). The exception is in cases where the IP stack interface is not able to indicate that the destination address was multicast.

The following are examples of valid query URIs:

- o ?href=/foo matches a link-value that is anchored at /foo
- o ?href=/foo\* matches a link-value that is anchored at a URI that starts with /foo
- o ?foo=bar matches a link value that has a target attribute named foo with the exact value bar
- o ?foo=bar\* matches a link value that has a target attribute named foo the value of which starts with bar, e.g., bar or barley
- o ?foo=\* matches a link value that has a target attribute named foo

## 5. Examples

A few examples of typical link descriptions in this format follows. Multiple resource descriptions in a representation are separated by commas. Linefeeds are also included in these examples for readability. Although the following examples use CoAP response codes, the examples are applicable to HTTP as well (the corresponding response code would be 200 OK).

This example includes links to two different sensors sharing the same Interface Description. Note that the default relation type for this link format is "hosts" in links with no rel= target attribute. Thus the links in this example tell that the Origin server /.well-known/core was requested from (the context) hosts the resources /sensors/temp and /sensors/light (each a target).

REQ: GET /.well-known/core

RES: 2.05 Content  
</sensors/temp>;if="sensor",  
</sensors/light>;if="sensor"

Without the linefeeds inserted here for readability, the format actually looks as follows.

```
</sensors/temp>;if="sensor",</sensors/light>;if="sensor"
```

This example arranges link descriptions hierarchically, with the entry point including a link to a sub-resource containing links about the sensors.

```
REQ: GET /.well-known/core
```

```
RES: 2.05 Content
</sensors>;ct=40
```

```
REQ: GET /sensors
```

```
RES: 2.05 Content
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor"
```

An example query filter may look like:

```
REQ: GET /.well-known/core?rt=light-lux
```

```
RES: 2.05 Content
</sensors/light>;rt="light-lux";if="sensor"
```

Note that relation-type attributes like `rt=`, `if=` and `rel=` can have multiple values separated by spaces. A query filter parameter can match any one of those values, as in this example:

```
REQ: GET /.well-known/core?rt=light-lux
```

```
RES: 2.05 Content
</sensors/light>;rt="light-lux core.sen-light";if="sensor"
```

This example shows the use of an anchor attribute to relate the temperature sensor resource to an external description and to an alternative URI.

REQ: GET /.well-known/core

RES: 2.05 Content

```
</sensors>;ct=40;title="Sensor Index",  
</sensors/temp>;rt="temperature-c";if="sensor",  
</sensors/light>;rt="light-lux";if="sensor",  
<http://www.example.com/sensors/t123>;anchor="/sensors/temp"  
;rel="describedby",  
</t>;anchor="/sensors/temp";rel="alternate"
```

If a client is interested to find relations about a particular resource, it can perform a query on the anchor parameter:

REQ: GET /.well-known/core?anchor=/sensors/temp

RES: 2.05 Content

```
<http://www.example.com/sensors/temp123>;anchor="/sensors/temp"  
;rel="describedby",  
</t>;anchor="/sensors/temp";rel="alternate"
```

The following example shows a large firmware resource with a size attribute. The consumer of this link would use the sz attribute to determine if the resource representation is too large and if block transfer would be required to request it. In this case a client with only a 64 KiB flash might only support a 16-bit integer for storing the sz attribute. Thus a special flag or value should be used to indicate "Big" (larger than 64 KiB).

REQ: GET /.well-known/core?rt=firmware

RES: 2.05 Content

```
</firmware/v2.1>;rt="firmware";sz=262144
```

## 6. Security Considerations

This specification has the same security considerations as described in Section 7 of [RFC5988]. The "/.well-known/core" resource MAY be protected e.g. using DTLS when hosted on a CoAP server as per [I-D.ietf-core-coap] Section 10.2.

Some servers might provide resource discovery services to a mix of clients that are trusted to different levels. For example, a lighting control system might allow any client to read state variables, but only certain clients to write state (turn lights on or

off). Servers that have authentication and authorization features SHOULD support authentication features of the underlying transport protocols (HTTP or DTLS/TLS) and allow servers to return different lists of links based on a client's identity and authorization. While such servers might not return all links to all requesters, not providing the link does not, by itself, control access to the relevant resource - a bad actor could know or guess the right URIs. Servers can also lie about the resources available. If it is important for a client to only get information from a known source, then that source needs to be authenticated.

Multicast requests using CoAP for the well-known link-format resources could be used to perform denial of service on a constrained network. A multicast request SHOULD only be accepted if the request is sufficiently authenticated and secured using e.g. IPsec or an appropriate object security mechanism.

CoRE link format parsers should be aware that a link description may be cyclical, i.e., contain a link to itself. These cyclical links could be direct or indirect (i.e., through referenced link resources). Care should be taken when parsing link descriptions and accessing cyclical links.

## 7. IANA Considerations

### 7.1. Well-known 'core' URI

This memo registers the "core" well-known URI in the Well-Known URI Registry as defined by [RFC5785].

URI suffix: core

Change controller: IETF

Specification document(s): [[ this document ]]

Related information: None

### 7.2. New 'hosts' relation type

This memo registers the new "hosts" Web Linking relation type as per [RFC5988].

Relation Name: hosts

Description: Refers to a resource hosted by the server indicated by the link context.

Reference: [[ this document ]]

Notes: This relation is used in CoRE where links are retrieved as a `"/.well-known/core"` resource representation, and is the default relation type in the CoRE Link Format.

Application Data: None

### 7.3. New link-format Internet media type

This memo registers the a new Internet media type for the CoRE link format, `application/link-format`.

Type name: `application`

Subtype name: `link-format`

Required parameters: None

Optional parameters: None

Encoding considerations: Binary data (UTF-8)

Security considerations:

Multicast requests using CoAP for the well-known link-format resources could be used to perform denial of service on a constrained network. A multicast request **SHOULD** only be accepted if the request is sufficiently authenticated and secured using e.g. IPsec or an appropriate object security mechanism.

CoRE link format parsers should be aware that a link description may be cyclical, i.e., contain a link to itself. These cyclical links could be direct or indirect (i.e., through referenced link resources). Care should be taken when parsing link descriptions and accessing cyclical links.

Interoperability considerations:

Published specification: [[ this document ]]

Applications that use this media type: CoAP server and client implementations for resource discovery and HTTP applications that use the link-format as a payload.

Additional information:

Magic number(s):

File extension(s): \*.wlnk

Macintosh file type code(s):

Intended usage: COMMON

Restrictions on usage: None

Author: CoRE WG

Change controller: IETF

#### 7.4. Constrained RESTful Environments (CORE) Parameters Registry

This specification establishes a new Constrained RESTful Environments (CORE) Parameters registry, which contains two new sub-registries of Link Target Attribute values (defined in [RFC5988]), one for Resource Type (rt=) Link Target Attribute values and the other for Interface Description (if=) Link Target Attribute values. No initial entries are defined by this specification for either sub-registry.

For both sub-registries, values starting with the characters "core" are registered using the IETF Review registration policy [RFC5226]. All other values are registered using the Specification Required policy, which requires review by a designated expert appointed by the IESG or their delegate.

The designated expert will enforce the following requirements:

- o Registration values MUST be related to the intended purpose of these attributes as described in Section 3.
- o Registered values MUST conform to the ABNF reg-rel-type definition of Section 2, meaning that the value starts with a lower case alphabetic character, followed by a sequence of lower case alphabetic, numeric, "." or "-" characters, and contains no white space.
- o It is recommended that the period "." character be used for dividing name segments, and that the dash "-" character be used for making a segment more readable. Example Interface Description values might be "core.batch" and "core.link-batch".
- o URIs are reserved for free use as extension values for these attributes, and MUST NOT be registered.

Registration requests consist of the completed registration template below, with the reference pointing to the required specification. To



allow for the allocation of values prior to publication, the designated expert may approve registration once they are satisfied that a specification will be published.

Note that link target attribute values can be registered by third parties, if the Designated Expert determines that an unregistered link target attribute values is widely deployed and not likely to be registered in a timely manner.

The registration template for both sub-registries is:

- o Attribute Value:
- o Description:
- o Reference:
- o Notes: [optional]

Registration requests should be sent to the [core-parameters@ietf.org](mailto:core-parameters@ietf.org) mailing list, marked clearly in the subject line (e.g., "NEW RESOURCE TYPE - example" to register an "example" relation type, or "NEW INTERFACE DESCRIPTION - example" to register an "example" interface description).

Within at most 14 days of the request, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

Decisions (or lack thereof) made by the Designated Expert can be first appealed to Application Area Directors (contactable using [app-ads@tools.ietf.org](mailto:app-ads@tools.ietf.org) email address or directly by looking up their email addresses on <http://www.iesg.org/> website) and, if the appellant is not satisfied with the response, to the full IESG (using the [iesg@iesg.org](mailto:iesg@iesg.org) mailing list).

## 8. Acknowledgments

Special thanks to Peter Bigot, who has made a considerable number reviews and text contributions that greatly improved the document. In particular, Peter is responsible for early improvements to the ABNF descriptions and the idea for a new "hosts" relation type.

Thanks to Mark Nottingham and Eran Hammer-Lahav for the discussions and ideas that led to this draft, and to Carsten Bormann, Martin

Thomson, Alexey Melnikov, Julian Reschke, Joel Halpern, Richard Barnes, Barry Leiba and Peter Saint-Andre for extensive comments and contributions that improved the text.

Thanks to Michael Stuber, Richard Kelsey, Cullen Jennings, Guido Moritz, Peter Van Der Stok, Adriano Pezzuto, Lisa Dussealt, Alexey Melnikov, Gilbert Clark, Salvatore Loreto, Petri Mutka, Szymon Sasin, Robert Quattlebaum, Robert Cragie, Angelo Castellani, Tom Herbst, Ed Beroaset, Gilman Tolle, Robby Simpson, Colin O'Flynn and David Ryan for helpful comments and discussions that have shaped the document.

## 9. Changelog

Changes from ietf-13 to ietf-14:

- o Editorial clarifications.
- o Examples and explanation for filtering when a target attribute of relation-type contains multiple values.

Changes from ietf-12 to ietf-13:

- o Improvements to the new CoRE Parameters registry
- o Replaced the Section 4.1 ABNF Query Filter definition with a URI Template (#240)
- o Aligned examples with rt= and if= value rules
- o Clarified that "href" can not be a link parameter

Changes from ietf-11 to ietf-12:

- o Changed "uri" to "href" in the filter query (#200)
- o Upgraded all ABNF to RFC5234 (#197)
- o Put multiple rt= and if= values in a single attribute (as in rel=) (#199)
- o Use the Origin definition (#191)
- o Clarified URI fetching rules (#196)
- o Added access control and other security consideration improvements (#189)

- o Fixed normalization for query pattern matching (#192)
- o Added an anchor restriction for hosts (#193)
- o New rules for determining link context (#194)
- o Described how to convert from HTTP Link Header (#190)
- o Created a registry for rt= and if= values (#195)
- o Integration of all other IETF LC and IESG comments.

Changes from ietf-10 to ietf-11:

- o Fixed editorial nits.

Changes from ietf-09 to ietf-10:

- o Changed to SHOULD NOT for multiple relation types (#178).
- o Changed to SHOULD NOT for multicast response repression (#179).
- o Updated ABNF for queries (#179).
- o Editorial improvements from WGLC comments.

Changes from ietf-08 to ietf-09:

- o Corrected ABNF and editorial nits.
- o Elided empty responses to multicast request.

Changes from ietf-07 to ietf-08:

- o IESG submission nits.

Changes from ietf-06 to ietf-07:

- o Moved the Content-type attribute (ct=) to the base CoAP specification.

Changes from ietf-05 to ietf-06:

- o Added improved text about the encoding of the format as UTF-8, but treating it as binary data without normalization.

Changes from ietf-04 to ietf-05:

- o Removed mention of UTF-8 as this is already defined by RFC5988 (#158)
- o Changed encoding considerations to "Binary data" (#157)
- o Updated ABNF to disallow leading zeros in integers (#159)
- o Updated examples and reference for coap-06 (#152)
- o Removed the application/link-format CoAP code registration, now included in the CoAP specification directly (#160)

Changes from ietf-03 to ietf-04:

- o Removed the attribute registry (#145).
- o Requested a CoAP media type for application/link-format (#144).
- o Editorial and reference improvements from AD review (#146).
- o Added a range limitation for ct attribute.
- o Added security considerations and file extension for application/link-format registration.

Changes from ietf-02 to ietf-03:

- o Removed 'obs' attribute definition, now defined in the CoAP Observation spec (#99).
- o Changed Resource name (n=) to Resource type (rt=) and d= to if= (#121).
- o Hierarchical organization of links under /.well-known/core removed (#95).
- o Bug in Section 3.1 on byte-wise query matching fixed (#91).
- o Explanatory text added about alternative Web link formats (#92).
- o Fixed a bug in Section 2.2.4 (#93).
- o Added use case examples (#89).
- o Clarified how the CoRE link format is used and how it differs from RFC5988 (#90, #98).

- o Changed the Interface definition format to quoted-string to match the resource type.
- o Added an IANA registry for CoRE Link Format attributes (#100).

Changes from ietf-01 to ietf-02:

- o Added references to RFC5988 (#41).
- o Removed sh and id link-extensions (#42).
- o Defined the use of UTF-8 (#84).
- o Changed query filter definition for any parameter (#70).
- o Added more example, now as a separate section (#43).
- o Mentioned cyclical links in the security section (#57).
- o Removed the sh and id attributes, added obs and sz attributes (#42).
- o Improved the context and relation description wrt RFC5988 and requested a new "hosts" default relation type (#85).

Changes from ietf-00 to ietf-01:

- o Editorial changes to correct references.
- o Formal definition for filter query string.
- o Removed URI-reference option from "n" and "id".
- o Added security text about multicast requests.

Changes from shelby-00 to ietf-00:

- o Fixed the ABNF link-extension definitions (quotes around URIs, integer definition).
- o Clarified that filtering is optional, and the query string is to be ignored if not supported (and the URI path processed as normally).
- o Required support of wildcard \* processing if filtering is supported.

- o Removed the assumption of a default content-type.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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, June 1999.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.
- [RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, August 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

### 10.2. Informative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",

draft-ietf-core-coap-09 (work in progress), March 2012.

- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>>.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.
- [RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.
- [WADL] Hadley, M., "Web Application Description Language (WADL)", 2009, <<http://java.net/projects/wadl/sources/svn/content/trunk/www/wadl20090202.pdf>>.

#### Author's Address

Zach Shelby  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: zach@sensinode.com





CoRE Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 13, 2012

K. Hartke  
Universitaet Bremen TZI  
March 12, 2012

Observing Resources in CoAP  
draft-ietf-core-observe-05

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. The state of a resource on a CoAP server can change over time. This document specifies a simple protocol extension for CoAP that gives clients the ability to observe such changes.

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 13, 2012.

Copyright Notice

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

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

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .  | 3  |
| 1.1. Background . . . . .  | 3  |
| 1.2. Protocol Overview . . . . .                                 | 3  |
| 1.3. Design Philosophy . . . . .                                 | 5  |
| 1.4. Conformance Requirements . . . . .                          | 6  |
| 2. The Observe Option . . . . .                                  | 6  |
| 3. Client-side Requirements . . . . .                            | 7  |
| 3.1. Request . . . . .   | 7  |
| 3.2. Notifications . . . . .                                     | 7  |
| 3.3. Caching . . . . .   | 8  |
| 3.4. Reordering . . . . .  | 9  |
| 3.5. Cancellation . . . . .                                      | 10 |
| 4. Server-side Requirements . . . . .                            | 10 |
| 4.1. Request . . . . .   | 10 |
| 4.2. Notifications . . . . .                                     | 11 |
| 4.3. Caching . . . . .   | 12 |
| 4.4. Reordering . . . . .  | 12 |
| 4.5. Retransmission . . . . .                                    | 13 |
| 5. Intermediaries . . . . .                                      | 13 |
| 6. Block-wise Transfers . . . . .                                | 14 |
| 7. Discovery . . . . .   | 15 |
| 8. Security Considerations . . . . .                             | 15 |
| 9. IANA Considerations . . . . .                                 | 16 |
| 10. Acknowledgements . . . . .                                   | 16 |
| 11. References . . . . .   | 16 |
| 11.1. Normative References . . . . .                             | 16 |
| 11.2. Informative References . . . . .                           | 17 |
| Appendix A. Examples . . . . .                                   | 18 |
| A.1. Proxying . . . . .  | 21 |
| A.2. Block-wise Transfer . . . . .                               | 23 |
| Appendix B. Modeling Resources to Tailor Notifications . . . . . | 23 |
| Appendix C. Changelog . . . . .                                  | 24 |
| Author's Address . . . . .                                       | 26 |

## 1. Introduction

### 1.1. Background

CoAP [I-D.ietf-core-coap] is an Application Protocol for Constrained Nodes/Networks. It is intended to provide RESTful services [REST] not unlike HTTP [RFC2616] while reducing the complexity of implementation as well as the size of packets exchanged in order to make these services useful in a highly constrained network of themselves highly constrained nodes.

The communication model of REST is that of a client exchanging resource representations with an origin server. The origin server is the definitive source for representations of the resources in its namespace. A client interested in a resource sends a request to the origin server that returns a response with a representation that is current at the time of the request.

This model does not work well when a client is interested in having a current representation of a resource over a period of time. Existing approaches when using HTTP, such as repeated polling or long-polls [RFC6202], generate significant complexity and/or overhead and thus are less applicable in a constrained environment.

The protocol specified in this document extends the CoAP core protocol with a mechanism to push resource representations from servers to interested clients, while still keeping the properties of REST.

Note that there is no intention for this mechanism to solve the full set of problems that the existing HTTP solutions solve, or to replace publish/subscribe networks that solve a much more general problem [RFC5989].

### 1.2. Protocol Overview

The protocol is based on the well-known observer design pattern [GOF].

In this design pattern, components, called `_observers_`, register at a specific, known provider, called the `_subject_`, that they are interested in being notified whenever the subject undergoes a change in state. The subject is responsible for administering its list of registered observers. If multiple subjects are of interest, an observer must register separately for all of them. The pattern is typically used when a clean separation between related components is required, such as data storage and user interface.

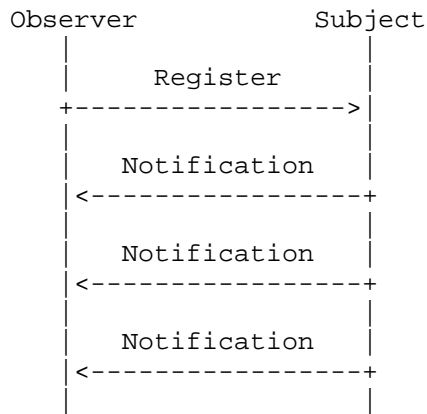


Figure 1: Observer Design Pattern

The observer design pattern is realized in CoAP as follows:

**Subject:** In the context of CoAP, the subject is a resource in the namespace of a CoAP server. The state of the resource can change over time, ranging from infrequent updates to continuous state transformations.

**Observer:** An observer is a CoAP client that is interested in the current state of the resource at any given time.

**Registration:** A client registers its interest by sending an extended GET request to the server. In addition to returning a representation of the target resource, this request causes the server to add the client to the list of observers of the resource.

**Notification:** Whenever the state of a resource changes, the server notifies each client registered as observer for the resource. Each notification is an additional CoAP response sent by the server in reply to the GET request and includes a complete representation of the new resource state.

Figure 2 shows an example of a CoAP client registering and receiving three notifications: the first upon registration and then two when the state of the resource changes. Registration request and notifications are identified by the presence of the Observe Option defined in this document. Notifications also echo the token specified by the client in the request, so the client can easily correlate them to the request.

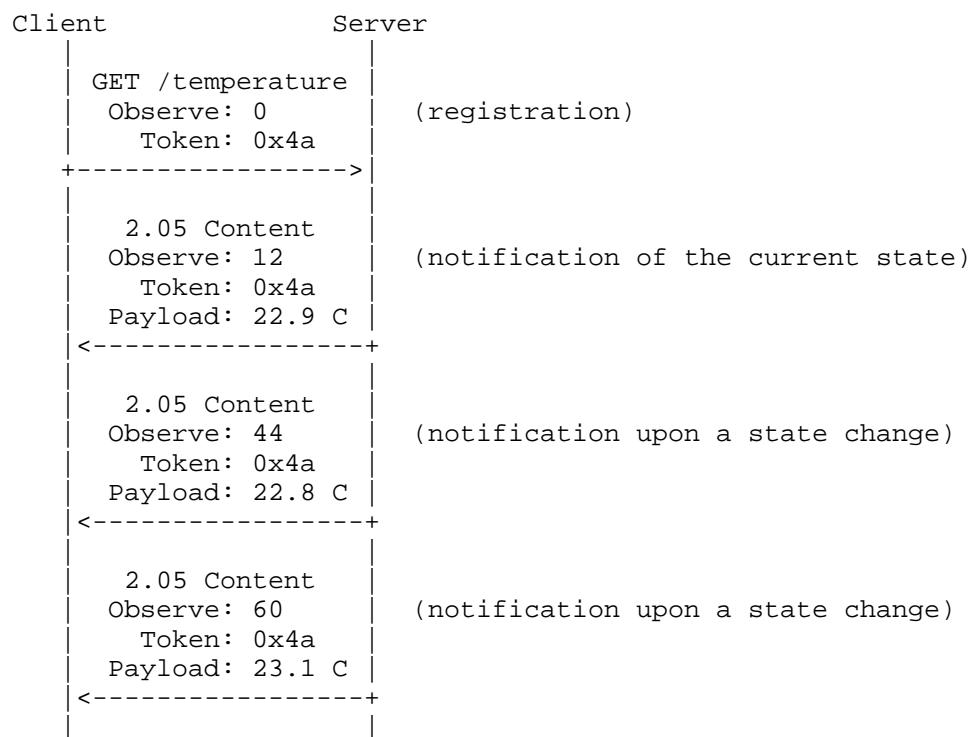


Figure 2: Observing a Resource in CoAP

The client is removed from the list of observers when it is no longer interested in the observed resource. The server can determine the client's continued interest from the client's acknowledgement of confirmable notifications. If a client wants to receive notifications after it has been removed from the list of observers, it needs to register again. The client can determine that it's still on the list of observers from the fact that it receives notifications. The protocol includes clear rules for what to do when a client does not receive a notification for some time, or a server does not receive acknowledgements.

### 1.3. Design Philosophy

The protocol builds on the architectural elements of REST, which include: a server that is responsible for the state and representation of the resources in its namespace, a client that is responsible for keeping the application state, and the stateless exchange of resource representations. (A server needs to keep track of the observers though, similar to how HTTP servers need to keep track of the TCP connections from their clients.) The protocol

enables high scalability and efficiency through the support of caches and intermediaries that multiplex the interest of multiple clients in the same resource into a single association.

The server is the authority for determining under what conditions resources change their state and how often observers are notified. The protocol does not offer explicit means for setting up triggers, thresholds or other conditions; it is up to the server to expose observable resources that change their state in a way that is meaningful for the application. Resources can be parameterized to achieve similar effects though; see Appendix B for examples.

Since bandwidth is in short supply in constrained environments, servers must adapt the rate of notifications to each client. This implies that a client cannot rely on observing every single state a resource goes through. Instead, the protocol is designed on the principle of eventual consistency: it guarantees that if the resource does not undergo a new change in state, eventually all observers will have a current representation of the last resource state.

#### 1.4. Conformance Requirements

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 [RFC2119].

## 2. The Observe Option

| No. | C/E      | Name    | Format | Length | Default |
|-----|----------|---------|--------|--------|---------|
| 10  | Elective | Observe | uint   | 0-2 B  | (none)  |

The Observe Option, when present, modifies the GET method so it does not only retrieve a representation of the current state of the resource identified by the request URI, but also requests the server to add the client to the list of observers of the resource. The exact semantics are defined in the sections below. The value of the option in a request MUST be zero on transmission and MUST be ignored on reception.

In a response, the Observe Option identifies the message as a notification, which implies that the client has been added to the list of observers and that the server will notify the client of further changes to the resource state. The option's value is a

sequence number that can be used for reordering detection (see Section 3.4 and Section 4.4). The value is encoded as a variable-length unsigned integer as defined in Appendix A of RFC XXXX [I-D.ietf-core-coap].

Since the Observe Option is elective, a GET request that includes the Observe Option will automatically fall back to a normal GET request if the server is unwilling or unable to add the client to the list of observers.

The Observe Option MUST NOT occur more than once in a request or response.

### 3. Client-side Requirements

#### 3.1. Request

A client can register its interest in a resource by issuing a GET request that includes an empty Observe Option. If the server returns a 2.xx response that includes an Observe Option as well, the server has added the client successfully to the list of observers of the target resource and the client will be notified of changes to the resource state for as long as the server can assume the client's interest.

#### 3.2. Notifications

Notifications are additional responses sent by the server in reply to the GET request. Each notification includes an Observe Option with a sequence number (see Section 3.4), a Token Option that matches the token specified by the client in the GET request, and a payload of the same media type as the initial response.

A notification can be confirmable or non-confirmable (i.e. sent in a confirmable or non-confirmable message). If a client does not recognize the token in a confirmable notification, it MUST NOT acknowledge the message and SHOULD reject it with a RST message. Otherwise, the client MUST acknowledge the message with an ACK message as usual. If a client does not recognize the token in a non-confirmable notification, it MAY reject it with a RST message.

An acknowledgement signals to the server that the client is alive and interested in receiving further notifications; if the server does not receive an acknowledgement in reply to a confirmable notification, it will assume that the client is no longer interested and will eventually remove it from the list of observers.

Notifications will have a 2.05 (Content) response code in most cases. They may also have a 2.03 (Valid) response code if the client includes an ETag Option in its request (see Section 3.3). In the event that the state of an observed resource is changed in a way that would cause a normal GET request to return an error (for example, when the resource is deleted), the server will send a notification with an error response code (4.xx/5.xx) and empty the list of observers of the resource.

### 3.3. Caching

As notifications are just additional responses, notifications partake in caching as defined by Section 5.6 of RFC XXXX [I-D.ietf-core-coap]. Both the freshness model and the validation model are supported. The freshness model also serves as the model for the client to determine if it's still on the list of observers or if it needs to re-register its interest in the resource.

A client MAY store a notification like a response in its cache and use a stored notification/response that is fresh without contacting the origin server. A notification/response is considered fresh while its age is not greater than its Max-Age and no newer notification has been received.

The server will do its best to keep the client up to date with a fresh representation of the current resource state. It will send a notification whenever the resource changes, or at latest when the age of the last notification becomes greater than its Max-Age. (Note that this notification may not arrive in time due to network latency.)

The client SHOULD assume that it's on the list of observers while the age of the last notification is not greater than Max-Age. If the client does not receive a notification before the age becomes greater than Max-Age, it can assume that it has been removed from the list of observers (e.g., due to a loss of server state). In this case, it may need to re-register its interest.

To make sure it has a fresh representation and/or to re-register its interest, a client MAY issue a new GET request with an Observe Option at any time. The GET request SHOULD specify a new token to avoid ambiguity, because the token serves as epoch identifier for the sequence numbers in the Observe Option (see Section 3.4).

It is RECOMMENDED that the client does not issue the request while it still has a fresh notification and, beyond that, while a new notification from the server is still likely to arrive. I.e. the client should wait until the age of the last notification becomes



greater than its Max-Age plus the potential retransmission window (see Section 4.1 of RFC XXXX [I-D.ietf-core-coap]) plus the expected maximum round trip time.

When a client has one or more notifications stored, it can use the ETag Option in the GET request to give the server an opportunity to select a stored response to be used. The client MAY include an ETag Option for each stored response that is applicable. It needs to keep those responses in the cache until it is no longer interested in receiving notifications for the target resource or it issues a new GET request with a new set of entity-tags. Whenever the observed resource changes its state to a representation identified by one of the ETag Options, the server can select a stored response by sending a 2.03 (Valid) notification with an appropriate ETag Option instead of a 2.05 (Content) notification.

### 3.4. Reordering

Messages that carry notifications can arrive in a different order than they were sent. Since the goal is eventual consistency (see Section 1.3), a client can safely skip a notification that arrives later than a newer notification. For this purpose, the server sets the value of the Observe Option in each notification to a sequence number.

A client MAY treat a notification as outdated (not fresh) under the following condition:

$$(V1 - V2) \% (2^{*16}) < (2^{*15}) \quad \text{and} \quad T2 < (T1 + (2^{*14}))$$

where V1 is the value of the Observe Option of the latest valid notification received, V2 the value of the Observe Option of the present notification, T1 a client-local timestamp of the latest valid notification received (in seconds), and T2 a client-local timestamp of the present notification.

Design Note: The first condition essentially verifies that  $V2 > V1$  holds in 16-bit sequence number arithmetic [RFC1982]. The second condition checks that the time expired between the two incoming messages is not so large that the sequence number might have wrapped around and the first check is therefore invalid. (In other words, after about  $2^{*14}$  seconds elapse without any notification, the client does not need to check the sequence numbers in order to assume an incoming notification is new.) The constants of  $2^{*14}$  and  $2^{*15}$  are non-critical, as is the even speed or precision of the clock involved.

### 3.5. Cancellation

When a client rejects a confirmable notification with a RST message or when it performs a GET request without an Observe Option for a currently observed resource, the server will remove the client from the list of observers for this resource. The client MAY use either method at any time to indicate that it is no longer interested in receiving notifications about a resource.

When a client rejects non-confirmable notification with a RST, there is also a chance that the server will remove the client from the list of observers for this resource. So the client MAY try this method as well. A client MAY rate-limit the RST messages it sends if the server appears to persistently ignore them.

Implementation Note: A client that does not mediate all its requests through its cache might inadvertently cancel an observation relationship by sending an unrelated GET to the same resource. To avoid this, without incurring a need for synchronization, such clients can use a different source transport address for these unrelated GET requests.

## 4. Server-side Requirements

### 4.1. Request

A GET request that includes an Observe Option requests the server not only to return a representation of the resource identified by the request URI, but also to add the client to the list of observers of the target resource. If no error occurs, the server MUST return a response with the representation of the current resource state and MUST notify the client of subsequent changes to the state as long as the client is on the list of observers.

A server that is unable or unwilling to add the client to the list of observers of the target resource MAY silently ignore the Observe Option and process the GET request as usual. The resulting response MUST NOT include an Observe Option, the absence of which signals to the client that it will not be notified of changes to the resource state and, e.g., needs to poll the resource instead.

If the client is already on the list of observers, the server MUST NOT add it a second time but MUST replace or update the existing entry. If the server receives a GET request that does not include an Observe Option, it MUST remove the client from the list of observers.

Two requests relate to the same list entry if both the request URI

and the source of the requests match. The source of a request is determined by the security mode used (see Section 10 of RFC XXXX [I-D.ietf-core-coap]): With NoSec, it is determined by the source IP address and UDP port number. With other security modes, the source is also determined by the security context. Note that Message IDs and Token Options MUST NOT be taken into account.

Any request with a method other than GET MUST NOT have a direct effect on a list of observers of a resource. However, such a request can have the indirect consequence of causing the server to send an error notification which does affect the list of observers (e.g., when a DELETE request is successful and an observed resource no longer exists).

#### 4.2. Notifications

A client is notified of a resource state change by an additional response sent by the server in reply to the GET request. Each such notification response MUST include an Observe Option and MUST echo the token specified by the client in the GET request. If there are multiple clients on the list of observers, the order in which they are notified is not defined; the server is free to use any method to determine the order.

A notification SHOULD have a 2.05 (Content) or 2.03 (Valid) response code. However, in the event that the state of a resource changes in a way that would cause a normal GET request to return an error (for example, if the resource is deleted), the server SHOULD notify the client by sending a notification with an appropriate error response code (4.xx/5.xx) and MUST empty the list of observers of the resource.

The media type used in a notification MUST be the same as the one used in the initial response to the GET request. If the server is unable to continue sending notifications using this media type, it SHOULD send a 5.00 (Internal Server Error) notification and MUST empty the list of observers of the resource.

A notification can be sent as a confirmable or a non-confirmable message. The message type used is typically application-dependent and MAY be determined by the server for each notification individually. For example, for resources that change in a somewhat predictable or regular fashion, notifications can be sent in non-confirmable messages; for resources that change infrequently, notifications can be sent in confirmable messages. The server can combine these two approaches depending on the frequency of state changes and the importance of individual notifications.

The acknowledgement of a confirmable notification implies the client's continued interest in being notified. If the client rejects a confirmable notification with a RST message, the server **MUST** remove the client from the list of observers. If the client rejects a non-confirmable notification with a RST message, the server **MAY** remove the client from the list of observers.

#### 4.3. Caching

The Max-Age Option of a notification **SHOULD** be set to a value that indicates when the server will send the next notification. For example, if the server sends a notification every 30 seconds, a Max-Age Option with value 30 should be included. The server **MAY** send a new notification before Max-Age ends and **MUST** send a new notification at latest when Max-Age ends. If the client does not receive a new notification before Max-Age ends, it will assume that it was removed from the list of observers (e.g., due to a loss of server state) and may issue a new GET request to re-register its interest.

It may not always be possible to predict when the server will send the next notification, for example, when a resource does not change its state in regular intervals. In this case, the server **SHOULD** set Max-Age to a good approximation. The value is a trade-off between increased usage of bandwidth and the risk of stale information. Smaller values lead to more notifications and more GET requests, while greater values result in network or device failures being detected later and data becoming stale.

The client can include a set of entity-tags in its request using the ETag Option. When the observed resource changes its state and the origin server is about to send a 2.05 (Content) notification, then, whenever that notification has an entity-tag in the set of entity-tags specified by the client, the server **MAY** send a 2.03 (Valid) response with an appropriate ETag Option instead. The server **MUST NOT** assume that the recipient has any response stored other than those identified by the entity-tags in the most recent GET request for the resource.

#### 4.4. Reordering

Because messages can get reordered, the client needs a way to determine if a notification arrived later than a newer notification. For this purpose, the server **MUST** set the value of the Observe Option in each notification to the 16 least-significant bits of a strictly increasing sequence number. The sequence number **MAY** start at any value. The server **MUST NOT** reuse the same option value with the same client, token and resource within approximately  $2^{16}$  seconds (roughly 18.2 hours).

Implementation Note: A simple implementation that satisfies the requirements is to use a timestamp (in seconds) provided by the device's clock, or a 16-bit unsigned integer variable that is incremented every second and wraps around every  $2^{16}$  seconds. It is not necessary that the clock reflects the correct local time or that it ticks exactly every second. Note that, on average, a server cannot send more than one notification per second per client, token and resource.

#### 4.5. Retransmission

In CoAP, confirmable messages are retransmitted in exponentially increasing intervals for a certain number of attempts until they are acknowledged by the client. In the context of observing a resource, it is undesirable to continue transmitting the representation of a resource state when the state has changed in the meantime.

When a server is in the process of delivering a confirmable notification and is waiting for an acknowledgement, and it wants to notify the client of a state change using a new confirmable message, it **MUST** stop retransmitting the old notification and **SHOULD** attempt to deliver the new notification with the number of attempts remaining from the old notification. When the last attempt to retransmit a confirmable message with a notification for a resource times out, the server **SHOULD** remove the client from the list of observers and **MAY** additionally remove the client from the lists of observers of all resources in its namespace.

The server **SHOULD** use a number of retransmit attempts (`MAX_RETRANSMIT`) such that removing a client from the list of observers before Max-Age ends is avoided.

A server **MAY** choose to skip a notification if it knows that it will send another notification soon (e.g., when the state is changing frequently). Similarly, it **MAY** choose to send a notification more than once. For example, when state changes occur in bursts, the server can skip some notifications, send the notifications in non-confirmable messages, and make sure that the client observes the latest state change after the burst by repeating the last notification in a confirmable message.

#### 5. Intermediaries

A client may be interested in a resource in the namespace of an origin server that is reached through one or more CoAP-to-CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the origin server, acting as if it was

communicating with the origin server itself as specified in Section 3. It is the task of this intermediary to provide the client with a current representation of the target resource and send notifications upon changes to the target resource state, much like an origin server as specified in Section 4.

To perform this task, the intermediary SHOULD make use of the protocol specified in this document, taking the role of the client and registering its own interest in the target resource with the next hop. If the next hop does not return a response with an Observe Option, the intermediary MAY resort to polling the next hop, or MAY itself return a response without an Observe Option. Note that the communication between each pair of hops is independent, i.e. each hop in the server role MUST determine individually how many notifications to send, of which type, and so on. Each hop MUST generate its own values for the Observe Option, and MUST set the value of the Max-Age Option according to the age of the local current representation.

Because a client (or an intermediary in the client role) can only be once in the list of observers of a resource at a server (or an intermediary in the server role) -- it is useless to observe the same resource multiple times -- an intermediary MUST observe a resource only once, even if there are multiple clients for which it observes the resource.

Note that an intermediary is not required to have a client to observe a resource; an intermediary MAY observe a resource, for instance, just to keep its own cache up to date.

See Appendix A.1 for examples.

## 6. Block-wise Transfers

Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. CoAP provides a block-wise transfer mechanism to address this problem [I-D.ietf-core-block]. The following rules apply to the combination of block-wise transfers with notifications.

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request. If the server supports block-wise transfers, it SHOULD take note of the block size for all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the server receives a new GET request from the client).

When sending a 2.05 (Content) notification, the server always sends

all blocks of the representation, suitably sequenced by its congestion control mechanism, even if only some of the blocks have changed with respect to a previous value. The server performs the block-wise transfer by making use of the Block2 Option in each block. When reassembling representations that are transmitted in multiple blocks, the client MUST NOT combine blocks carrying different Observe Option values, or blocks that have been received more than approximately  $2^{*}14$  seconds apart.

See Appendix A.2 for an example.

## 7. Discovery

A web link [RFC5988] to a resource accessible by the CoAP protocol MAY indicate that the server encourages the observation of this resource by including the target attribute "obs". This is particularly useful in link-format documents [I-D.ietf-core-link-format].

This target attribute is used as a flag, and thus it has no value component -- a value given for the attribute MUST NOT be given for this version of the specification and MUST be ignored if present. The target attribute "obs" MUST NOT be given more than once for this version of the specification.

## 8. Security Considerations

The security considerations of RFC XXXX [I-D.ietf-core-coap] apply.

Note that the considerations about amplification attacks are somewhat amplified when observing resources. In NoSec mode, a server MUST therefore strictly limit the number of notifications that it sends between receiving acknowledgements that confirm the actual interest of the client in the data; i.e., any notifications sent in non-confirmable messages MUST be interspersed with confirmable messages. (An attacker may still spoof the acknowledgements if the confirmable messages are sufficiently predictable.)

As with any protocol that creates state, attackers may attempt to exhaust the resources that the server has available for maintaining the list of observers for each resource. Servers MAY want to access-control this creation of state. As degraded behavior, the server can always fall back to processing the request as a normal GET request (without an Observe Option) if it is unwilling or unable to add a client to the list of observers of a resource, including if system resources are exhausted or nearing exhaustion.

Intermediaries MUST be careful to ensure that notifications cannot be employed to create a loop. A simple way to break any loops is to employ caches for forwarding notifications in intermediaries.

## 9. IANA Considerations

The following entries are added to the CoAP Option Numbers registry:

| Number | Name    | Reference |
|--------|---------|-----------|
| 10     | Observe | [RFCXXXX] |

## 10. Acknowledgements

Carsten Bormann was an original author of this draft and is acknowledged for significant contribution to this document.

Thanks to Daniele Alessandrelli, Jari Arkko, Peter Bigot, Angelo Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Cullen Jennings, Matthias Kovatsch, Salvatore Loreto, Charles Palmer and Zach Shelby for helpful comments and discussions that have shaped the document.

Klaus Hartke was funded by the Klaus Tschira Foundation.

## 11. References

### 11.1. Normative References

- [I-D.ietf-core-block]  
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-08 (work in progress), February 2012.
- [I-D.ietf-core-coap]  
Frank, B., Bormann, C., Hartke, K., and Z. Shelby, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-08 (work in progress), October 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.



## 11.2. Informative References

- [GOF] Gamma, E., Helm, R., Johnson, R., and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, USA, November 1994.
- [I-D.ietf-core-link-format] Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-11 (work in progress), January 2012.
- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <<http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>>.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.
- [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, June 1999.
- [RFC5989] Roach, A., "A SIP Event Package for Subscribing to Changes to an HTTP Resource", RFC 5989, October 2010.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.

## Appendix A. Examples

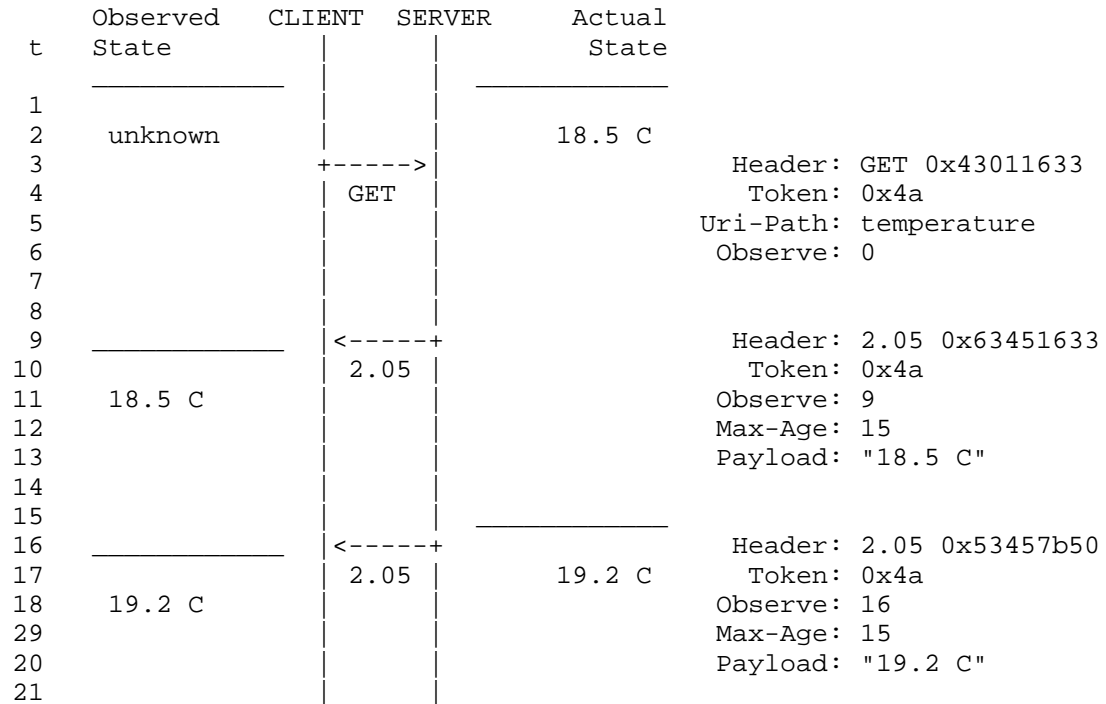


Figure 3: A client registers and receives a notification of the current state and upon a state change

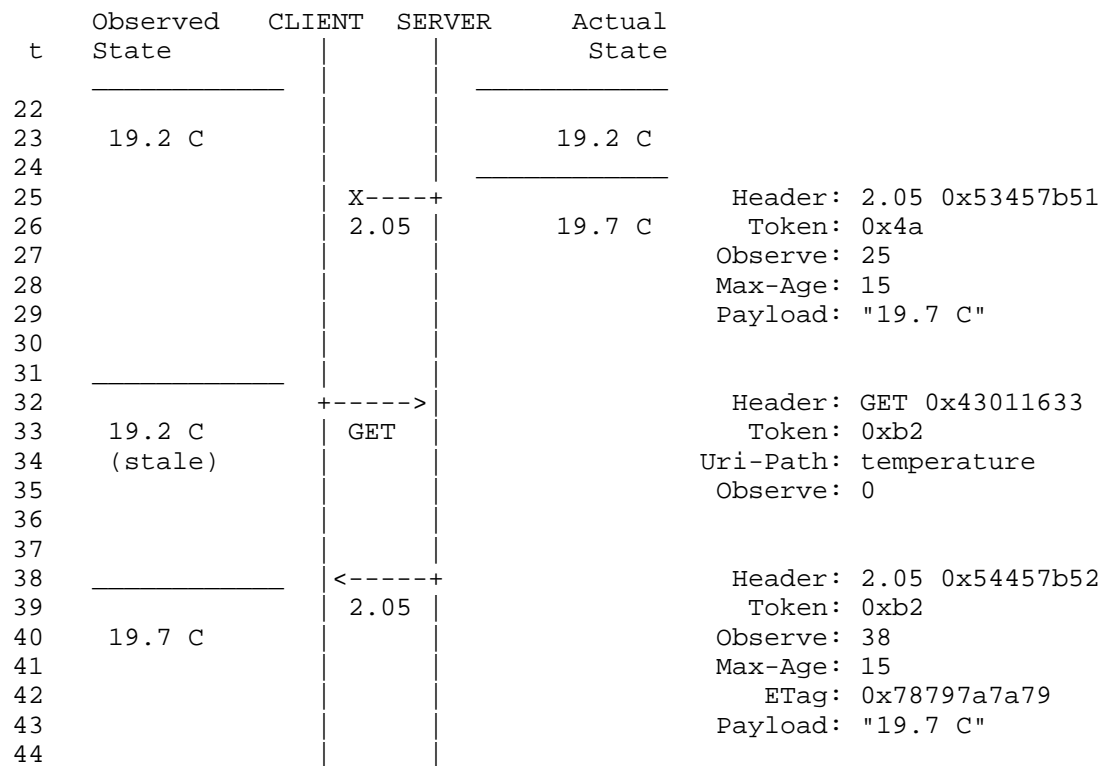


Figure 4: The client re-registers after Max-Age ends

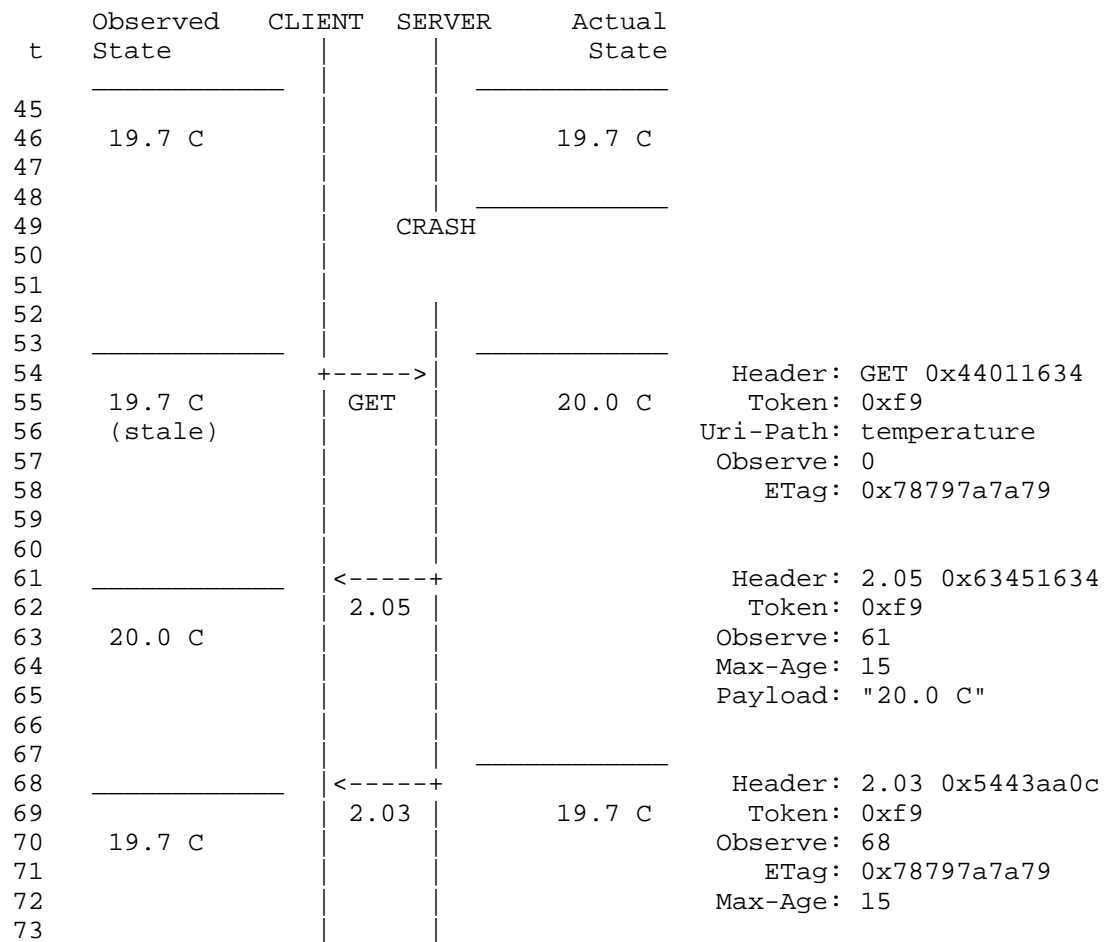


Figure 5: The client re-registers and gives the server the opportunity to select a stored response

## A.1. Proxying

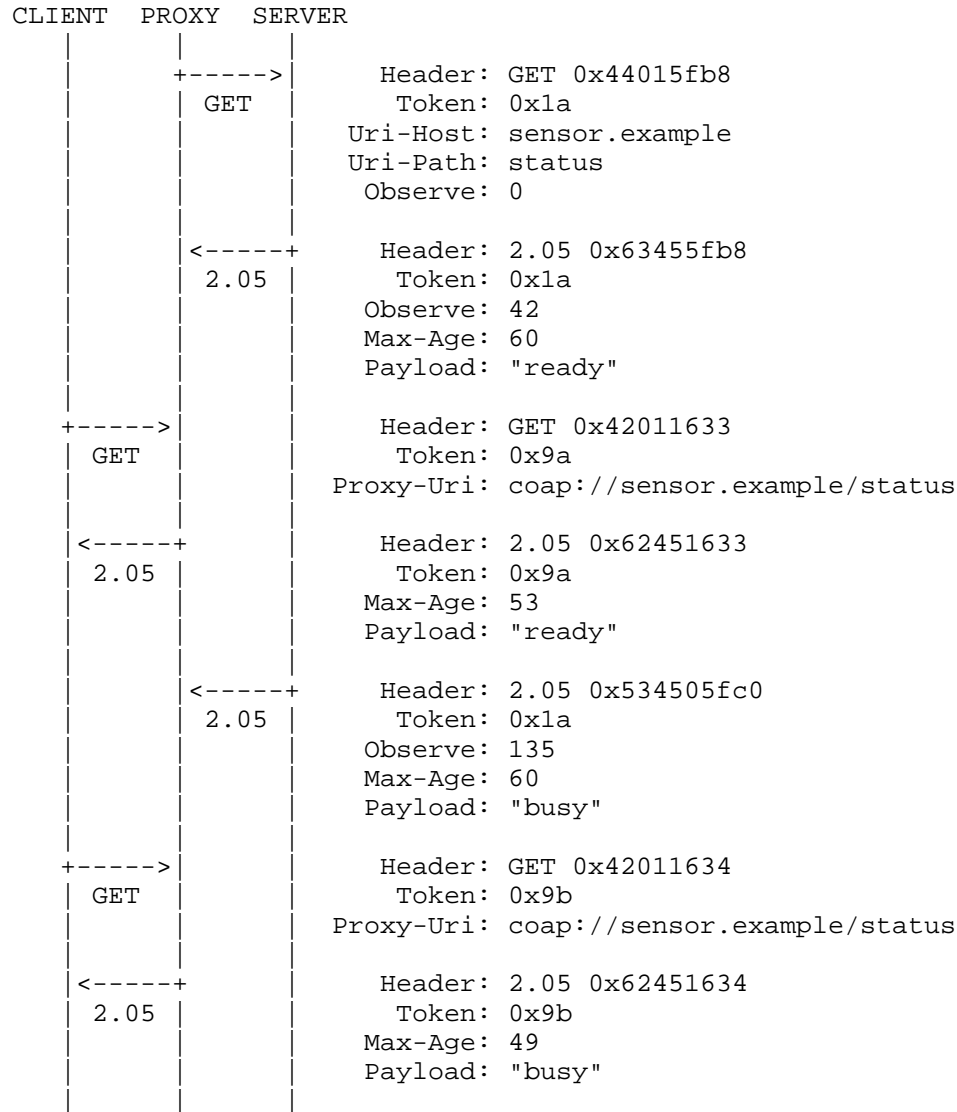


Figure 6: A proxy observes a resource to keep its cache up to date

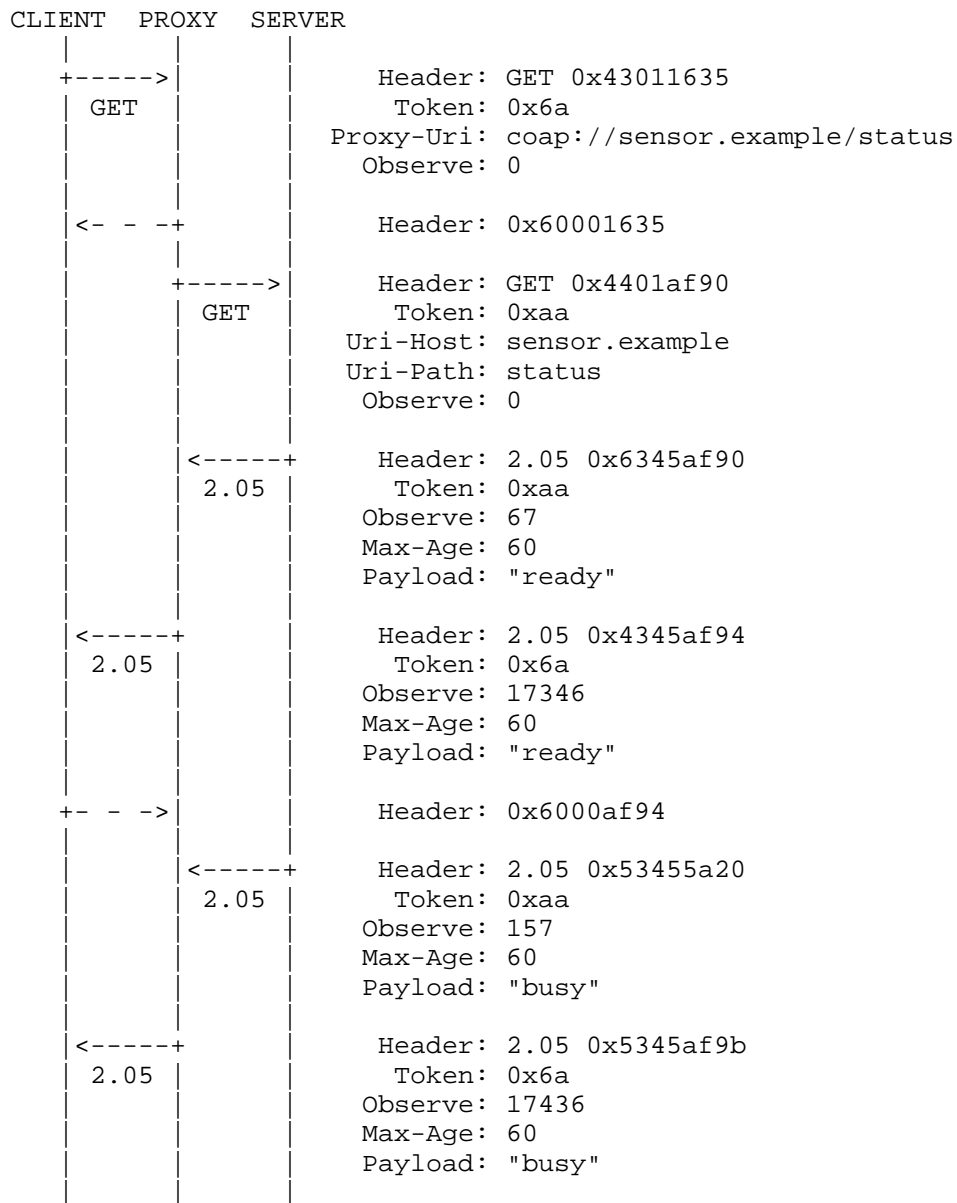


Figure 7: A client observes a resource through a proxy

## A.2. Block-wise Transfer

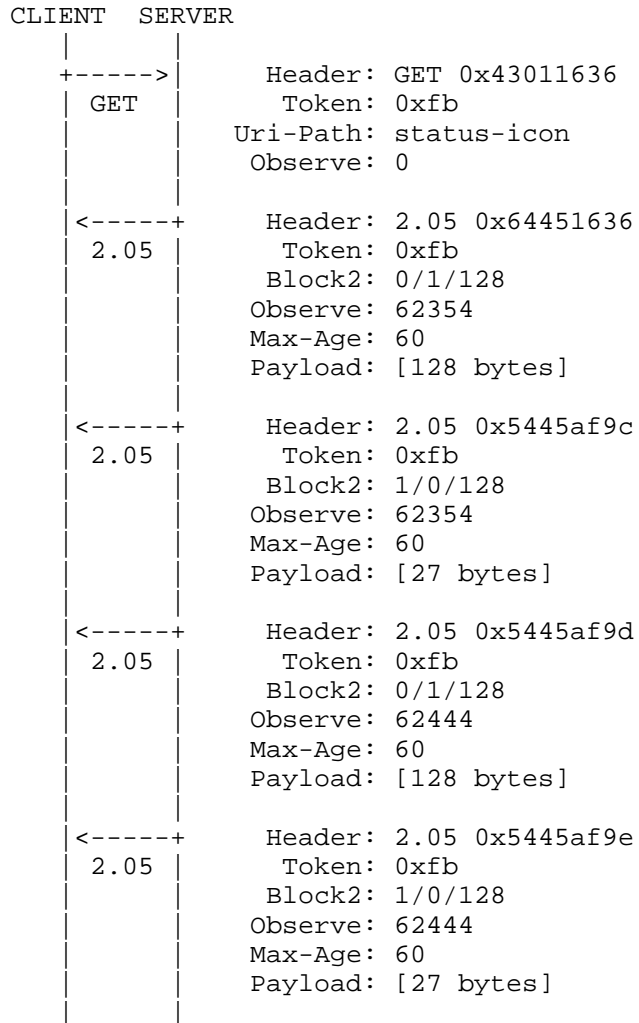


Figure 8: A server sends two notifications of two blocks each

## Appendix B. Modeling Resources to Tailor Notifications

A server may want to provide notifications that respond to very specific conditions on some state. This is best done by modeling the resources that the server exposes according to these needs.

For example, for a CoAP server with an attached temperature sensor,

- o the server could, in the simplest form, expose a resource `<coap://server/temperature>` that changes its state every second to the current temperature measured by the sensor;
- o the server could, however, also expose a resource `<coap://server/temperature/felt>` that changes its state to "cold" when the temperature drops below a preconfigured threshold, and to "warm" when the temperature exceeds a second, higher threshold;
- o the server could expose a parameterized resource `<coap://server/temperature/critical?above=45>` that changes its state to the current temperature if the temperature exceeds the specified value, and changes its state to "OK" when the temperature drops below; or
- o the server could expose a parameterized resource `<coap://server/temperature?query=select+avg(temperature)+from+Sensor.window:time(30sec)>` that accepts expressions of arbitrary complexity and changes its state accordingly.

In any case, the client is notified about the current state of the resource whenever the state of the appropriately modeled resource changes. By designing resources that change their state on certain conditions, it is possible to notify the client only when these conditions occur instead of continuously supplying it with information it doesn't need. With parametrized resources, this is not limited to conditions defined by the server, but can be extended to arbitrarily complex conditions defined by the client. Thus, the server designer can choose exactly the right level of complexity for the application envisioned and devices used, and is not constrained to a "one size fits all" mechanism built into the protocol.

## Appendix C. Changelog

Changes from ietf-04 to ietf-05:

- o Recommended that a client does not re-register while a new notification from the server is still likely to arrive. This is to avoid that the request of the client and the last notification after max-age cross over each other (#174).
- o Relaxed requirements when sending RST in reply to non-confirmable notifications.
- o Added an implementation note about careless GETs (#184).



- o Updated examples.

Changes from ietf-03 to ietf-04:

- o Removed the "Max-OFE" Option.
- o Allowed RST in reply to non-confirmable notifications.
- o Added a section on cancellation.
- o Updated examples.

Changes from ietf-02 to ietf-03:

- o Separated client-side and server-side requirements.
- o Fixed uncertainty if client is still on the list of observers by introducing a liveliness model based on Max-Age and a new option called "Max-OFE" (#174).
- o Simplified the text on message reordering (#129).
- o Clarified requirements for intermediaries.
- o Clarified the combination of block-wise transfers with notifications (#172).
- o Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
- o Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

- o Removed the requirement of periodic refreshing (#126).
- o The new "Observe" Option replaces the "Lifetime" Option.
- o Introduced a new mechanism to detect message reordering.
- o Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- o Changed terminology from "subscriptions" to "observation relationships" (#33).

- o Changed the name of the option to "Lifetime".
- o Clarified establishment of observation relationships.
- o Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- o Clarified rules for establishing observation relationships (#68).
- o Clarified conditions under which an observation relationship is terminated.
- o Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).
- o Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).
- o Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- o Added a mechanism to detect message reordering (#35).
- o Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).
- o Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- o Described interaction with block-wise transfers (#36).
- o Added Resource Discovery section (#99).
- o Added IANA Considerations.
- o Added Security Considerations (#40).
- o Added examples (#38).

Author's Address

Klaus Hartke  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63905  
Fax: +49-421-218-7000  
Email: hartke@tzi.org



CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: June 19, 2015

K. Li  
R. Sun  
Huawei Technologies  
December 16, 2014

CoAP Payload-Length Option Extension  
draft-li-core-coap-payload-length-option-03

Abstract

This document defines an extension to the Constrained Application Protocol (CoAP) to add one new option: Payload-Length, which is used to indicate the length of the payload of the message.

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 June 19, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

|                                      |   |
|--------------------------------------|---|
| 1. Introduction . . . . .            | 2 |
| 1.1. Motivation . . . . .            | 2 |
| 1.2. Justification . . . . .         | 2 |
| 1.3. Terminology . . . . .           | 3 |
| 1.4. Requirements Language . . . . . | 3 |
| 2. Option Definition . . . . .       | 3 |
| 3. Example . . . . .                 | 4 |
| 4. IANA Considerations . . . . .     | 4 |
| 5. Security Considerations . . . . . | 4 |
| 6. Acknowledgements . . . . .        | 5 |
| 7. Normative References . . . . .    | 5 |
| Authors' Addresses . . . . .         | 5 |

## 1. Introduction

This specification adds one new option to the Constrained Application Protocol (CoAP): Payload-Length.

## 1.1. Motivation

If a CoAP message is transported through UDP, the message length can be obtained from the UDP header. But not all transport mechanisms provide an unambiguous length of the CoAP message. For example, in industry field, there are some data transport protocols, like RS232, RS422, RS485, which don't provide message length indication. For these cases, an indication of the payload length of the message is needed in CoAP message level.

TBD: how about CAN bus protocol, USB 2.0?

With this option, it will be easier for the receiver to extract the payload part from the whole message.

Another benefit to have this option is to check the integrity of the message length.

## 1.2. Justification

To indicate the payload length, another alternative is to use encoding method as specified in section 3.2 of [RFC7252], but it is better to use an Option for this.

Reason is that, payload length is an optional feature, and in most of the cases, it is not necessary to be indicated. If we use encoding method, every implementation needs to support this encoding for the

payload, not only for the options. If we use an Option for this, it is optional, and it can be optionally implemented where necessary.

### 1.3. Terminology

The terms CoAP Server and CoAP Client are used synonymously to Server and Client as specified in the terminology section of [RFC7252].

### 1.4. 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].

## 2. Option Definition

| Type | C | U | N | R | Name           | Format | Length | Default |
|------|---|---|---|---|----------------|--------|--------|---------|
| TBD  | - | - | - | - | Payload-Length | uint   | 0-2 B  | (none)  |

If this option is present, the value of this option is an unsigned integer giving the length of the payload of the message. Note that this integer can be zero for a zero-length payload, which can in turn be represented by a zero-length option value.

The Payload-Length option does not have a default value, so in case of its absence the receiver MUST determine the payload length through other means. This is to keep backward compatibility. If the option is absent, the payload can have any size, and the payload length needs to be determined as it is currently done for UDP.

The minimum payload length is 0, and the maximum payload length is  $2^{16}-1=65535$ .

In case that the transport layer does not provide message length indication, the Payload-Length option SHOULD be included in the CoAP message. Otherwise, it MAY be included.

This options can be used both in the request and response.

This option MUST NOT occur more than once.

### 3. Example

In the example below, in the GET request, the payload is empty, so the Payload-Length option has a zero-length option payload. In the response, the payload is "22.3 C", and the Payload-Length is 6.

```

Client  Server
|      |
|      |
+-----+ Header: GET (T=CON, Code=1, MID=0x7d38)
| GET   | Token: 0x53
|      | Uri-Path: "temperature"
|      | Payload-Length: 0
|      |
|- - - + Header: (T=ACK, Code=0, MID=0x7d38)
|      |
|      |
<-----+ Header: 2.05 Content (T=CON, Code=69, MID=0xad7b)
| 2.05 | Token: 0x53
|      | Payload: "22.3 C"
|      | Payload-Length: 6
|      |
+- - -> Header: (T=ACK, Code=0, MID=0xad7b)
|      |
|      |

```

### 4. IANA Considerations

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

| Number | Name           | Reference                  |
|--------|----------------|----------------------------|
| TBD    | Payload-Length | Section 2 of this document |

### 5. Security Considerations

The Payload-Length option defined in this document presents no security considerations beyond those in Section 10 of the base CoAP specification [RFC7252].



## 6. Acknowledgements

The authors of this draft would like to thank Carsten Bormann and Klaus Hartke, for the initial texts in draft [I-D.bormann-coap-misc].

The authors of this draft would like to thank Bert Greevenbosch and Xianghui Sun for the discussion and review.

## 7. Normative References

- [I-D.bormann-coap-misc]  
Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-27 (work in progress), November 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

## Authors' Addresses

Kepeng Li  
Huawei Technologies  
Huawei Base, Bantian, Longgang District  
Shenzhen, Guangdong 518129  
P. R. China

Phone: +86-755-289718087  
Email: likepeng@huawei.com

Ruinan Sun  
Huawei Technologies  
Huawei Base, Bantian, Longgang District  
Shenzhen, Guangdong 518129  
P. R. China

Phone: +86-755-28970171  
Email: sunruinan@huawei.com

core  
Internet-Draft  
Intended status: Standards Track  
Expires: April 18, 2015

S. Li  
K. Li  
Huawei Technologies  
J. Hoebeke  
F. Van den Abeele  
iMinds-IBCN/UGent  
A. Jara  
University of Murcia  
October 15, 2014

Conditional observe in CoAP  
draft-li-core-conditional-observe-05

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. Through the Observe option, clients can observe changes in the state of resources and obtain a current representation of the last resource state. This document defines two new options for CoAP Observe so that a CoAP client can specify timing conditions when observing a resource on a CoAP server. As a result, the CoAP client is only informed about resource state changes when the timing conditions are met. This offers possibilities to extend network intelligence, enhance scalability, and optimize the lifetime and performance in order to address the requirements from the Constrained Nodes and Networks.

Note

Discussion and suggestions for improvement are requested, and should be sent to [core@ietf.org](mailto:core@ietf.org).

Status of This Memo

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

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

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

This Internet-Draft will expire on April 18, 2015.

#### Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

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

#### Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .                | 2  |
| 1.1. Justification . . . . .             | 3  |
| 1.2. Terminology . . . . .               | 4  |
| 2. Motivation . . . . .                  | 4  |
| 3. Conditional Observe Options . . . . . | 5  |
| 4. Using the Condition Option . . . . .  | 6  |
| 5. Examples . . . . .                    | 7  |
| 6. Security Considerations . . . . .     | 10 |
| 7. IANA Considerations . . . . .         | 10 |
| 8. Acknowledgements . . . . .            | 11 |
| 9. References . . . . .                  | 11 |
| 9.1. Normative References . . . . .      | 11 |
| 9.2. Informative References . . . . .    | 11 |
| Appendix A. Change log . . . . .         | 12 |
| Authors' Addresses . . . . .             | 13 |

#### 1. Introduction

CoAP [RFC7252] is an Application Protocol for Constrained Nodes/Networks. The observe [I-D.ietf-core-observe] specification describes a protocol design so that a CoAP client and server can use the subject/observer design pattern to establish an observation relationship. When observe is used, the CoAP client will get a notification response whenever the state of the observed resource changed. However, in some scenarios, the CoAP client may only be interested in the state changes of the resource after a specified time interval, to avoid superfluous traffic. This memo defines two new CoAP options "Minimum-Interval" and "Maximum-Interval" that can be used to allow the CoAP client to set conditions for the

observation relationship, and only when such condition is met, the CoAP server will send the notification response with the latest state change. When such a condition fails, the CoAP server does not need to send the notification response.

### 1.1. Justification

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

CoAP is used for Constrained Networks, especially used for transporting sensor data. Different sensor equipments have different properties, e.g. different change rates, different response time, etc. resulting in varying clients' interests that differ from mere state changes. As such, when a client wants to collect information from a sensor, it does not want to receive too many notification messages within a short time period, if the state of a server resource changes too often. Also, if the resource's representation does not change for a long time, the client wants to receive notifications in order to make sure that the observe relationship is still alive.

Consider the following example.

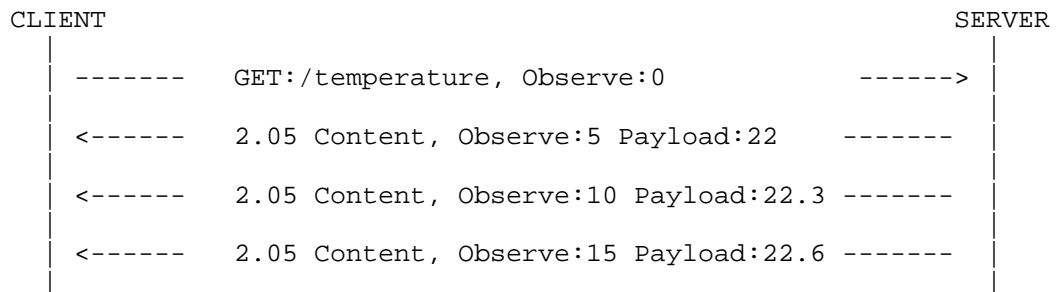


Figure 1: GET request with observe

In this example, the sensor acts as a server, and it collects the resource data every 5 seconds. When the client observes a resource on the server, it will receive a response whenever the server updates the resource, that is to say, mostly every 5 seconds the client will receive a notification response. However, the client might be a

simple constrained device not too sensitive to the resource state change, so it may not want to receive notifications that often. One possible solution could be to alter the sensor's configuration, e.g. to shorten the collecting frequency. However, the sensor should be able to provide services to many other clients, making it hard to find the best configuration that fits all clients' requirements.

## 1.2. Terminology

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

## 2. Motivation

The CoAP Observe Option gives clients the ability to observe changes in the state of resources. An observe relationship is established and whenever the state of the resource changes, the new representation is pushed to the observer. In some cases, the server resource changes too often, while the client does not want to receive notifications that often. The client just wants to receive notifications after a specific time interval.

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

In [I-D.ietf-core-interfaces], a mechanism is described to provide additional information to the Observe Option through the use of query parameters. It is possible to define a fixed set of query parameters to enable conditional observations. However, many more query parameters can be offered by a resource for different purposes. This complicates the automatic processing of conditional observations. Also embedding the query parameters in the URI encumbers processing at intermediaries. To alleviate this problem, this draft proposes to use CoAP options to specify timing-related conditions, that is, minimum time interval and maximum time interval, for the notifications. Using options ensures a compact representation and well-defined meaning. For resource value related conditions, e.g. larger than, smaller than, another mechanism such as query parameters

can be used to complement the Minimum-Interval and Maximum-Interval Options.

### 3. Conditional Observe Options

| No. | C | U | N | R | Name             | Format | Length | Default |
|-----|---|---|---|---|------------------|--------|--------|---------|
| TBD |   | x | - |   | Minimum-Interval | int    | 0-2 B  | (none)  |
| TBD |   | x | - |   | Maximum-Interval | int    | 0-2 B  | (none)  |

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Figure 2: Conditional Observe Options

This draft defines two condition options for observe, Minimum-Interval and Maximum-Interval. Both options are elective, and Proxy Unsafe (similar to the Observe Option).

The Minimum-Interval and Maximum-Interval options can only be present in a GET request message and its response message. They must be used together with the Observe Option, since they extend the meaning of the Observe Option. The Minimum-Interval/Maximum-Interval MUST be included in the first notification message if the conditional observation relationship is created successfully. If the server does not support the Minimum-Interval/Maximum-Interval option contained in the observe request, it will ignore the them.

The Minimum-Interval Option indicates the minimum time interval between two notification responses sent from the server, even if the resource representation changes. After sending the previous notification response, the server has to wait for the minimum time interval to expire, before sending the subsequent notification response, if the resource representation changes within the specified minimum time interval.

The Maximum-Interval Option indicates that the maximum time interval between two notification responses sent from the server, even if the resource representation does not change. After sending the previous notification response, if the resource representation does not change after the maximum time interval, the server needs to send the same resource representation immediately after timeout of the maximum time interval. This can be used to show the liveness of the server.

Basically, a similar procedure as described in the observe draft [I-D.ietf-core-observe] is followed, but extended with additional

behavior by taking the Minimum-Interval and Maximum-Interval Options into account. The exact semantics are described below.

The values of the Minimum-Interval and Maximum-Interval Options are measured in seconds. The range is from 1 second to  $2^{16}$  seconds, that is, 65256 seconds, around 18 hours. There is no default value for the Minimum-Interval/Maximum-Interval option.

The Minimum-Interval Option and Maximum-Interval Option are elective, and may be present separately or together. If both of the options are included in a request, their relationship is "AND", meaning that the server will only send a notification when both conditions are fulfilled. Note that if both options are present, Maximum-Interval option should be greater than or equal to Minimum-Interval option. If the Minimum-Interval Option and Maximum-Interval Option are both present and have the same value, then the server should send notifications periodically, regardless whether the resource representation has changed or not.

#### 4. Using the Condition Option

Whenever a client wants to initiate a Conditional Observation relationship, it sends a GET request with both an Observe option and at least one Conditional Observe Option. The Conditional Observe Options represent the minimum and maximum time interval conditions the client wants to apply to the observation relationship.

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

Since the Minimum-Interval/Maximum-Interval option is elective, an observe request that includes the option will automatically fall back to a basic observe request if the server does not support the Minimum-Interval/Maximum-Interval option. This implies that the client will now receive notifications as described in [I-D.ietf-core-observe] and that the client itself is responsible for

processing the resource state in the notifications in order to identify whether the condition of interest is fulfilled or not.

Whenever the state of a resource that supports conditional observations changes on the server, the server needs to check the established conditional relationships. Whenever the relationship condition(s) is(are) met, the server sends the notification response to the client that has established the relationship. In case the server is still waiting for a confirmable notification to be acknowledged or the 3 seconds on average for a non-confirmable notification to elapse, it MUST adhere to the behaviour specified in section 4.5 of [I-D.ietf-core-observe].

## 5. Examples

This section gives some short examples with message flows to illustrate the use of the Minimum-Interval and Maximum-Interval Options in an observe GET request.

The following example shows that the client sets the Minimum-Interval Option to 10 seconds, This means that the server shall wait at least 10s between sending notification responses, indicating changes in the state of the resource, to the client.



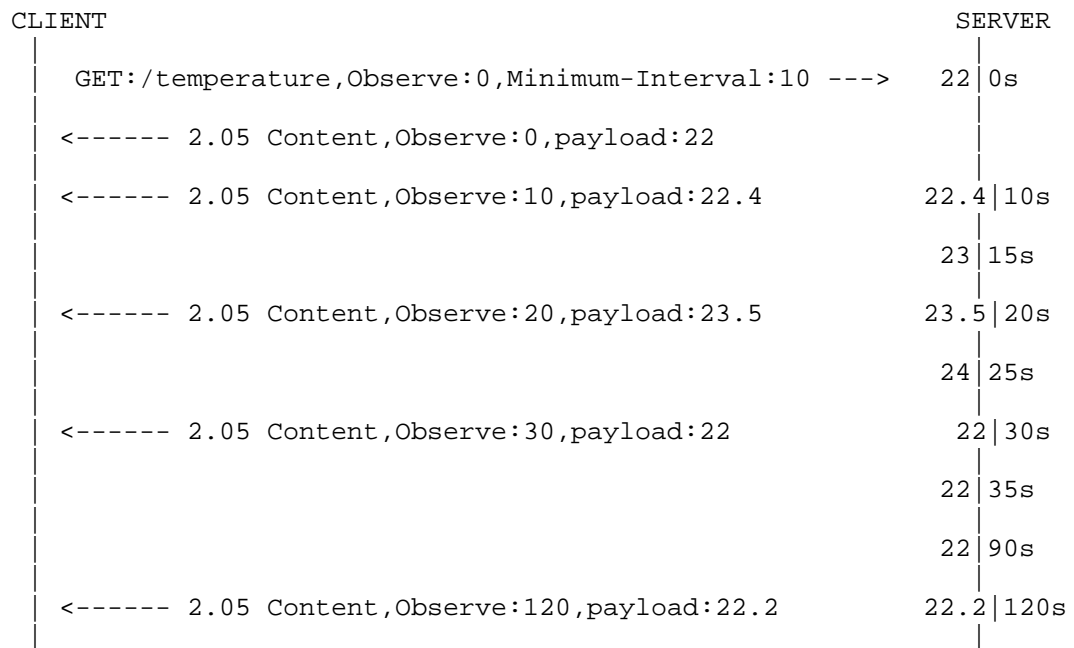


Figure 3: Minimum-Interval Option Usage

The next example shows that the client sets the Maximum-Interval Option to 60 seconds. The server will send notifications upon every state change, but will leave maximally 60s between subsequent notifications, even if they do not incur a state change.

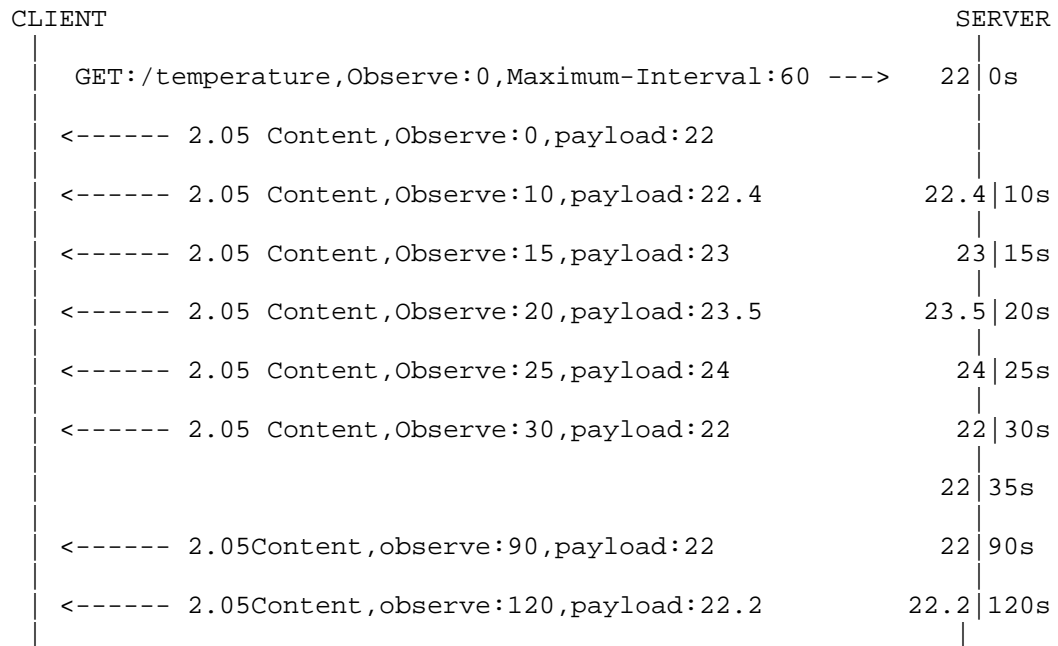


Figure 4: Maximum-Interval Option Usage

The next example shows a client that sets both the Minimum-Interval Option and the Maximum-Interval Option to 30 seconds. As a result, the server sends notifications every 30 seconds, independent of whether the resource has changed or not.

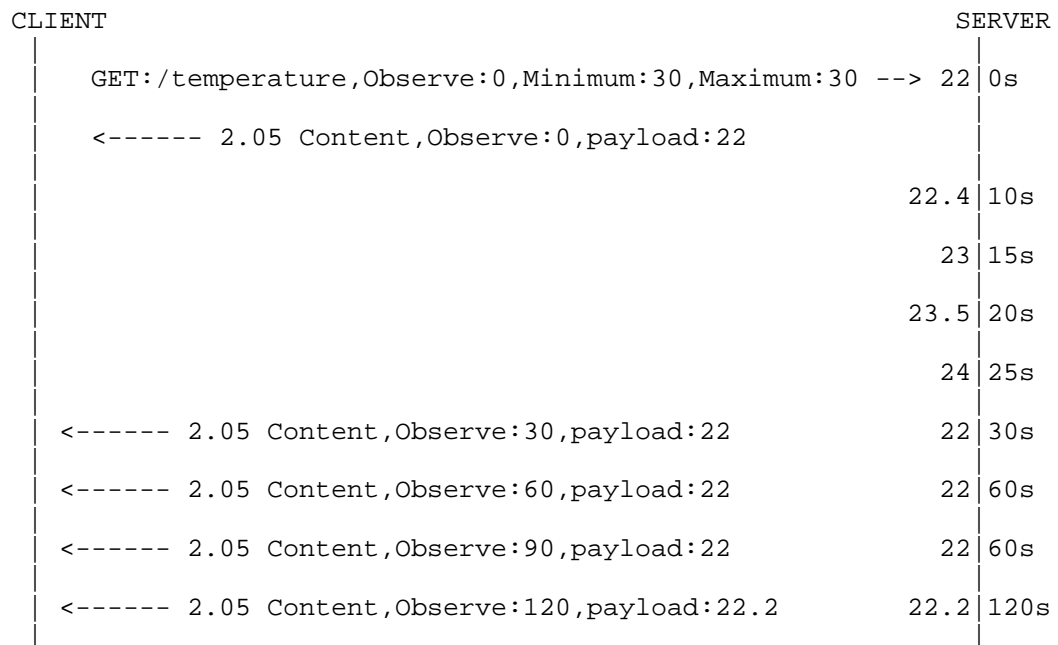


Figure 5: Minimum-Interval and Maximum-Interval Options together

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

## 6. Security Considerations

As the Minimum-Interval and Maximum-Interval options are used together with the Observe option, when it is used it must follow the security considerations as described in Observe draft [I-D.ietf-core-observe].

## 7. IANA Considerations

This draft adds the following option numbers to the CoAP Option Numbers registry of [RFC7252]

| Number | Name             | Reference |
|--------|------------------|-----------|
| TBD    | Minimal-Interval | [RFCXXXX] |
| TBD    | Maximum-Interval | [RFCXXXX] |

Table 3: Conditional Observe Option Numbers

## 8. Acknowledgements

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

Thanks to Zach Shelby, Bert Greevenbosch for the discussions.

## 9. References

### 9.1. Normative References

- [I-D.ietf-core-observe] Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-14 (work in progress), June 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

### 9.2. Informative References

- [CPSCOM] Ketema, G., Hoebeke, J., Moerman, I., Demeester, P., Li, Shi., and A. Jara, "Efficiently observing Internet of Things Resources", The 2012 IEEE International Conference on Cyber, Physical and Social Computing November 20-23, 2012, Besancon, France, Novemer 2012.
- [I-D.bormann-coap-misc] Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-13 (work in progress), March 2012.
- [I-D.ietf-core-interfaces] Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-core-interfaces-01 (work in progress), December 2013.

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

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

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

## Appendix A. Change log

### Changes in v05

- o Simplified to contain only two options:

- \* Minimal Interval
- \* Maximum Interval

### Changes in v04

- o Updated draft to be consistent with observe draft:
  - \* (Took request URI into consideration for Cancellation.)
  - \* (Explicitly stated that the draft diverts from a MUST NOT defined in the observe draft)
  - \* Stated that a server should follow the text from the observe draft for an unacknowledged notification in regards to the transmission of new notifications and the cancellation of existing relationships.
  - \* Removed obsolete Pledge and Keep-Alive options due to decoupling the freshness of a representation from whether or not the client is still on the list of observers in observe v08. Instead, rely on server-side initiated Confirmable messages (as defined in the Observe) for checking the existence of a relationship.
- o (Switched Reliability flag with Logic flag.)
- o Updated source endpoint terminology.

### Changes in v03

- o Examples for most condition types
- o Update the option number according to the new numbering scheme

- o Added reference to paper validating implementation on constrained device

#### Changes in v02

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

#### Authors' Addresses

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

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

Kepeng Li  
Huawei Technologies  
Huawei Base  
Bantian, Longgang  
Shenzhen, Guangdong 518129  
China

Phone: +86-755-28970231  
Email: likepeng@huawei.com

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

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

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

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

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

CORE WG  
Internet-Draft  
Intended status: Standards Track  
Expires: August 16, 2014

A. Rahman  
InterDigital Communications, LLC  
February 12, 2014

Enhanced Sleepy Node Support for CoAP  
draft-rahman-core-sleepy-05

Abstract

CoAP is a specialized web transfer protocol for constrained devices. These devices typically have some combination of limited battery power, small memory footprint and low throughput links. It is expected that in CoAP networks there will be a certain portion of devices that are "sleepy" and which may occasionally go into a sleep mode (i.e. go into a low power state to conserve power) and temporarily suspend CoAP protocol communication. This document proposes a minimal and efficient mechanism building on the Resource Directory (RD) functionality to enhance sleepy node support in CoAP networks. The RD functionality may be incorporated as part of a CoAP Proxy.

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 August 16, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of



publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|  |    |
|--|----|
| 1. Terminology and Conventions . . . . .                       | 2  |
| 2. Introduction . . . . .                                      | 3  |
| 3. Proposal . . . . .  | 4  |
| 3.1. RD Based Sleep Tracking . . . . .                         | 4  |
| 3.2. Example of Synchronous RD Based Sleep Tracking . . . . .  | 5  |
| 3.3. Example of Asynchronous RD Based Sleep Tracking . . . . . | 7  |
| 3.4. RD Caching Proxy . . . . .                                | 11 |
| 4. Performance Results . . . . .                               | 11 |
| 4.1. Prototype Setup . . . . .                                 | 11 |
| 4.2. Initial Results . . . . .                                 | 13 |
| 4.3. Comparison with Observe . . . . .                         | 13 |
| 5. Acknowledgements . . . . .                                  | 14 |
| 6. IANA Considerations . . . . .                               | 14 |
| 7. Security Considerations . . . . .                           | 14 |
| 8. References . . . . .  | 14 |
| 8.1. Normative References . . . . .                            | 14 |
| 8.2. Informative References . . . . .                          | 14 |
| Author's Address . . . . .                                     | 15 |

## 1. Terminology and Conventions

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 [RFC2119].

This document assumes readers are familiar with the terms and concepts that are used in [I-D.ietf-core-coap] and [RFC6690]. In addition, this document defines the following terminology:

### Sleepy Node

A sleepy node is a CoAP client or server that may sometimes go into a sleep mode (i.e. go into a low power state to conserve power) and temporarily suspend CoAP protocol communication. A sleepy node will otherwise remain in a fully powered on state where it has the capability to perform full CoAP protocol communication.

### Non-Sleepy Node

A non-sleepy node is a CoAP client or server that always remains in a fully powered on state (i.e. always awake) where it has the capability to perform full CoAP protocol communication. The general operation of non-sleepy nodes is assumed to be well known and so are not explicitly spelled out in this document except where needed for clarity.

## 2. Introduction

The current CoAP approach provides a minimal support of sleepy nodes as follows:

- o [I-D.ietf-core-coap] defines CoAP proxies which can cache and service requests for sleepy CoAP servers. A client explicitly sends a CoAP request (GET) to a forward proxy (identified by its IP address) while indicating the URI (of the resource of interest) associated to a sleepy CoAP origin server. If the proxy has a valid representation of the resource in its cache it can then respond directly to the client regardless of the current sleep state of the origin server. Otherwise the proxy has to attempt to retrieve (GET) the resource from the sleepy origin server. The attempt may or may not be successful depending on the sleep state of the origin server.
- o [RFC6690] and [I-D.ietf-core-resource-directory] defines a Resource Directory (RD) mechanism where sleepy CoAP servers can register/update (POST/PUT to `"/.well-known/core"`) their list of resources on a central (non-sleepy) RD server. This allows clients to discover the list of resources from the RD (GET `/rd-lookup/...`) for a sleepy server, regardless of its current sleep state. Unlike a proxy, the RD stores only the URIs for other nodes, and not the actual resource representation [RFC6690]. The client then may attempt to operate on (GET/PUT/POST/DELETE) the desired resource at the sleepy origin server. This attempt may or may not be successful depending on the sleep state of the origin server.
- o Lower layer (i.e. below the IP layer) support for sleepy nodes exist in most wireless technologies (e.g. IEEE 802.11 (WiFi), and IEEE 802.15.4 (ZigBee)). For example, most wireless technologies support limited functionality such as packet scheduling to account for sleepy nodes in their physical and MAC layer protocols. This lower layer functionality is not aware of any specific timing or operational aspects of application layer protocols like CoAP.

Some more background information regarding Sleepy Nodes can be found in [I-D.rahman-core-sleepy-nodes-do-we-need].

### 3. Proposal

#### 3.1. RD Based Sleep Tracking

The current CoAP approach to support sleepy nodes can be significantly improved by introducing RD based mechanisms for a CoAP client to determine whether:

- o A targeted resource is located on a sleepy server.
- o A sleepy server is currently in sleep mode or not.

We define the following new parameters to characterize a sleepy node:

- o SleepState - Indicates whether the node is currently in sleep mode or not (i.e. Sleeping or Awake).
- o SleepDuration - Indicates the maximum duration of time that the node stays in sleep mode.
- o TimeSleeping - Indicates the length of time the node has been sleeping (i.e. if Sleep State = Sleeping).
- o NextSleep - Indicates the next time the node will go to sleep (i.e. if Sleep State = Awake).

These parameters are all server (node) level and are new parameters added to the RD URI Template Variables defined in [I-D.ietf-core-resource-directory].

We also define a new lookup-type ("ss") for the RD lookup interface specified in [I-D.ietf-core-resource-directory]. This new lookup-type supports looking up the SleepState of a specified end-point.

The three time based parameters (SleepDuration, TimeSleeping, NextSleep) can be based on either an absolute network time (for a time synchronized network) or a relative local time (measured at the local node).

Following the approach of [RFC6690] and [I-D.ietf-core-resource-directory], sleep parameters for sleepy servers can be stored by the server in the RD and accessed by all interested clients. Examples of using these parameters in a synchronous or asynchronous manner are shown in the following sections.

### 3.2. Example of Synchronous RD Based Sleep Tracking

Figure 1 shows an example of using RD based sleep tracking in a synchronous fashion:

(1) SleepyNode-1 is awake and having previously discovered the local RD, stores its CORE link format in the RD (POST/rd) identified by its entry point (?ep=SleepyNode-1). The sleep parameters are also updated as part of this step.

(2)-(3) RD services the POST and stores the CORE link format and starts the sleep timers for this node.

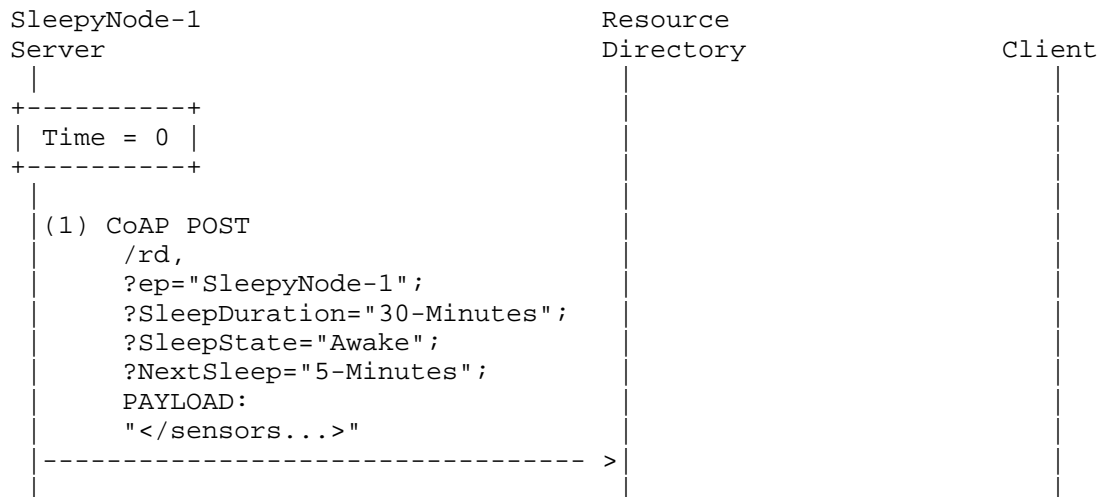
(4) SleepyNode-1 falls asleep.

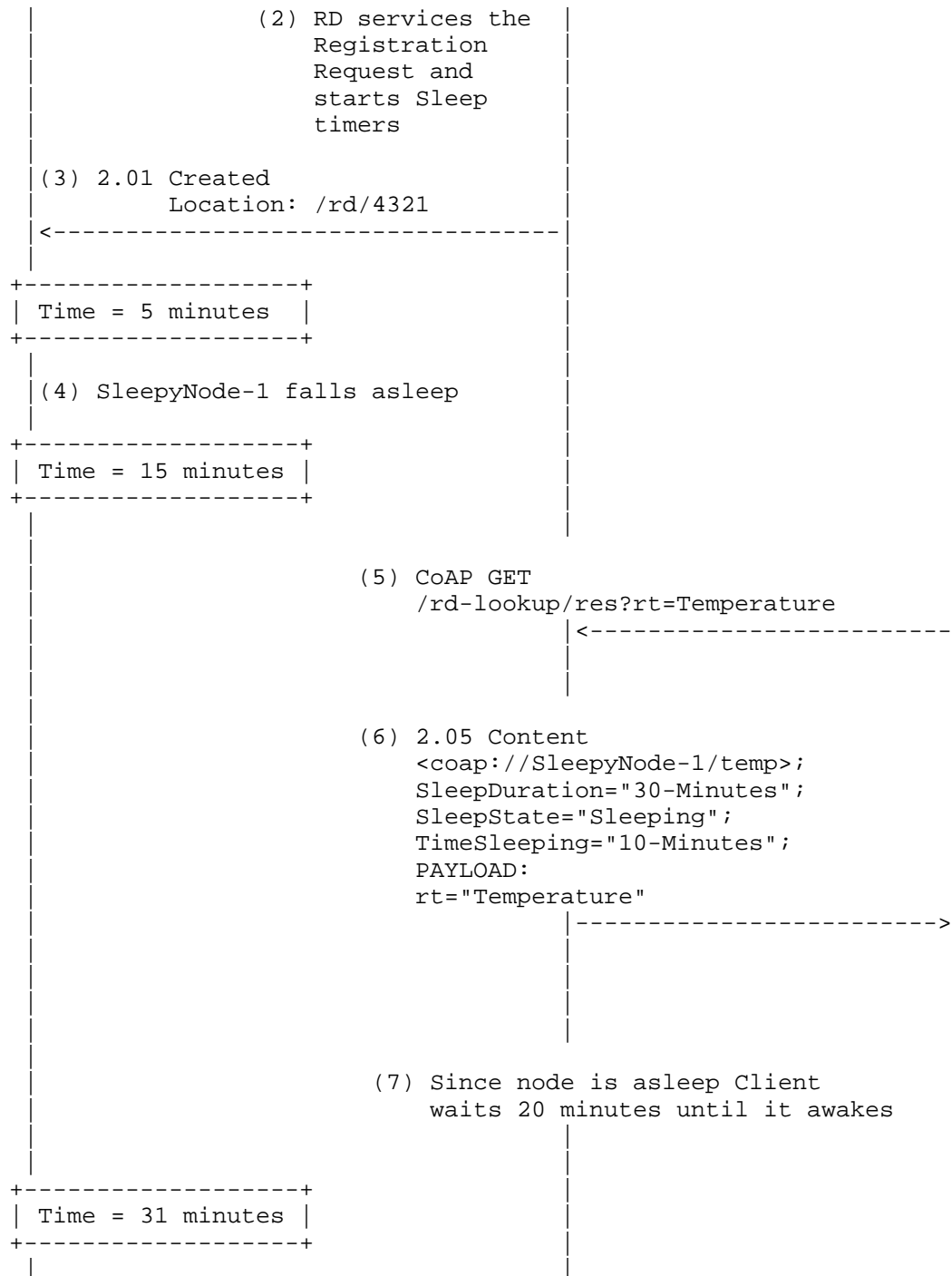
(5) A client is interested in temperature sensors in this domain and does a lookup on the RD.

(6) The RD does a lookup and finds that SleepyNode-1 is the only node meeting the match and sends back the required info including the Sleep parameters.

(7) From the sleep parameters, the client knows that the node is currently asleep and so internally schedules to send the GET request when the node wakes up (plus a small safety hysteresis).

(8)-(9) Client sends GET request for temperature sensors and successfully receives the content as SleepyNode-1 is awake.





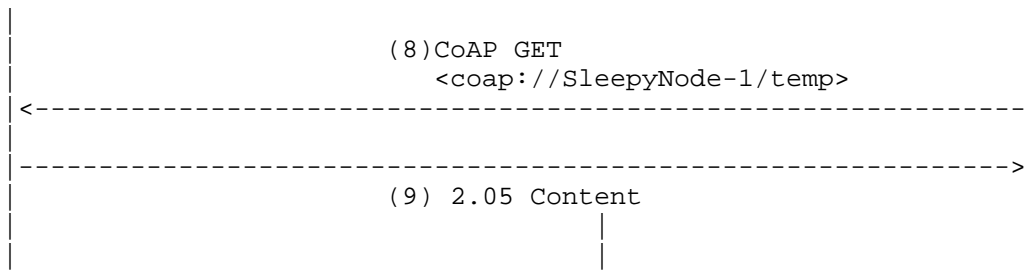


Figure 1: Synchronous Resource Directory Based Sleep Tracking

### 3.3. Example of Asynchronous RD Based Sleep Tracking

Figure 2 shows an example of using RD based sleep tracking in an asynchronous fashion:

(1) SleepyNode-1 is awake and having previously discovered the local RD, stores its CORE link format in the RD (POST/rd) identified by its entry point (?ep=SleepyNode-1).

(2)-(3) RD services the POST and stores the CORE link format.

(4) A client is interested in temperature sensors in this domain and does a lookup on the RD for all sensors that are currently awake.

(5) The RD does a lookup and finds that SleepyNode-1 is the only node meeting the match and sends back the required info.

(6)-(7) Using the sleep state lookup functionality (lookup-type := "ss"), the client adds itself to the list of observers to get SleepState updates from RD for SleepyNode-1 [I-D.ietf-core-observe].

(8)-(9) Client performs RD 'resource' lookup to find URI of temperature sensor of resource hosted on SleepyNode-1.

(10)-(13) SleepyNode-1 prepares to go to sleep and updates the SleepState in the RD.

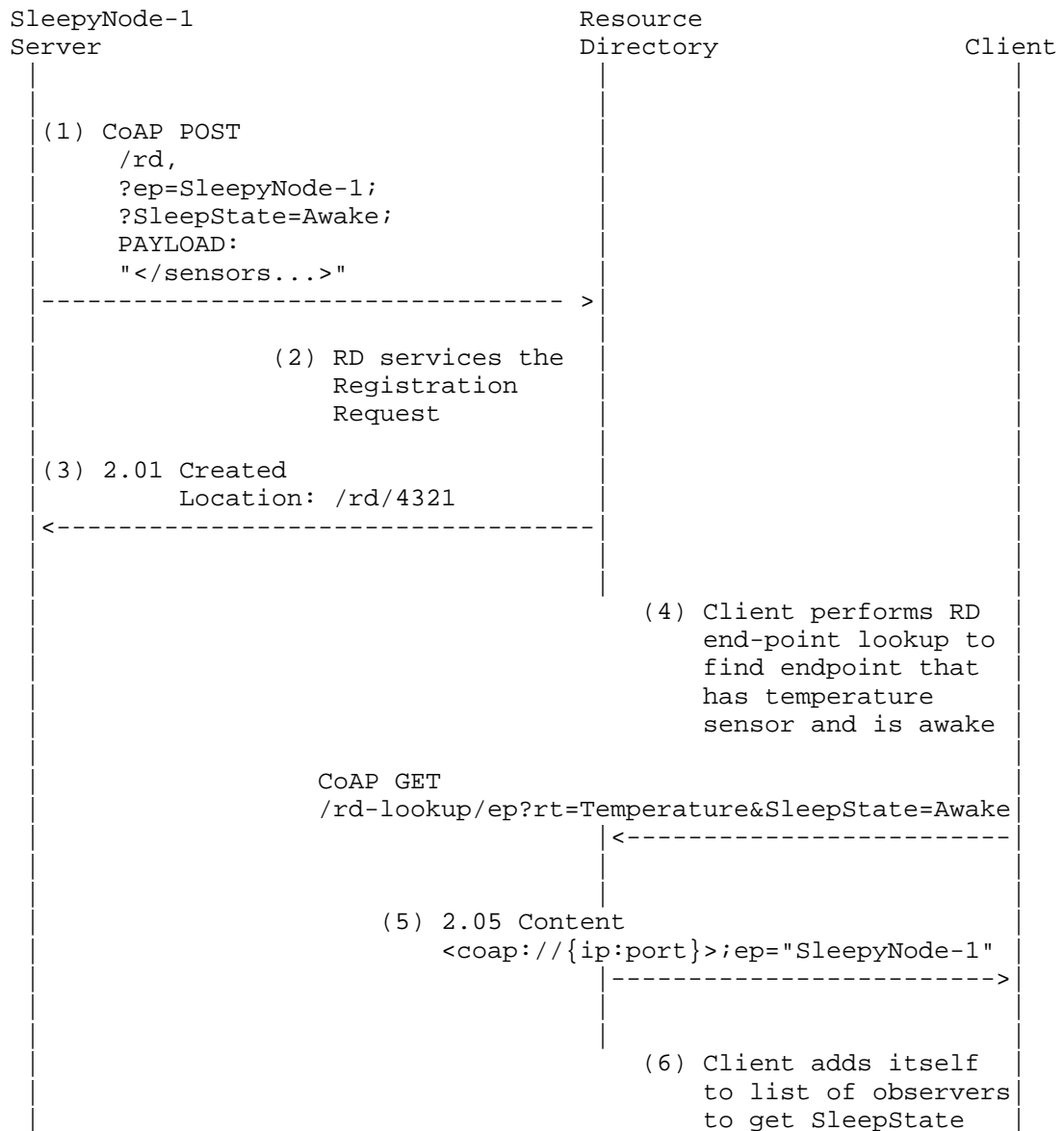
(14) RD notifies the client through previously established observe relationship.

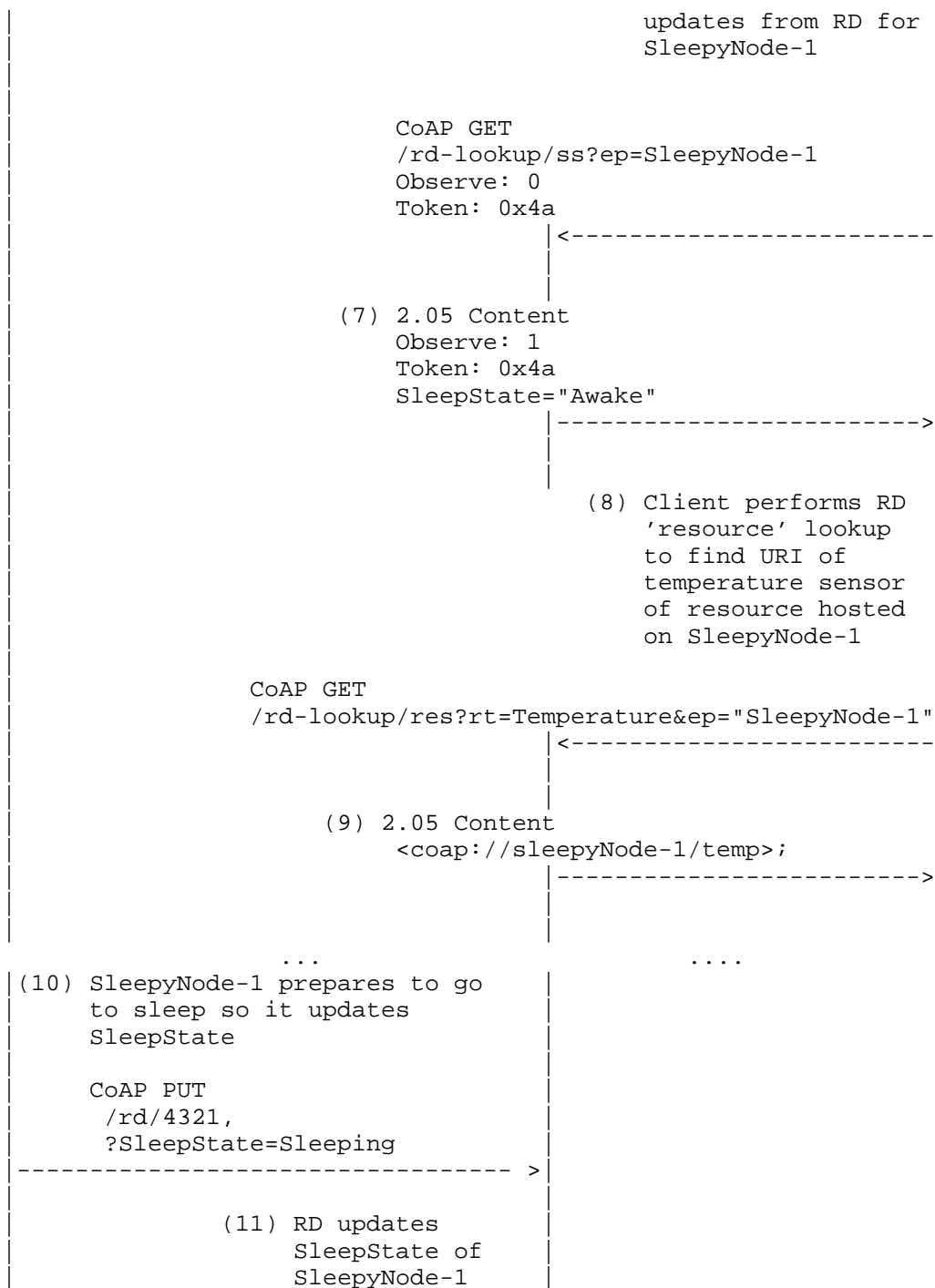
(15) Client application wants to get the temperature now but does not send the request as it knows SleepyNode-1 is currently sleeping.

(16)-(19) SleepyNode-1 wakes up and updates the SleepState in the RD.

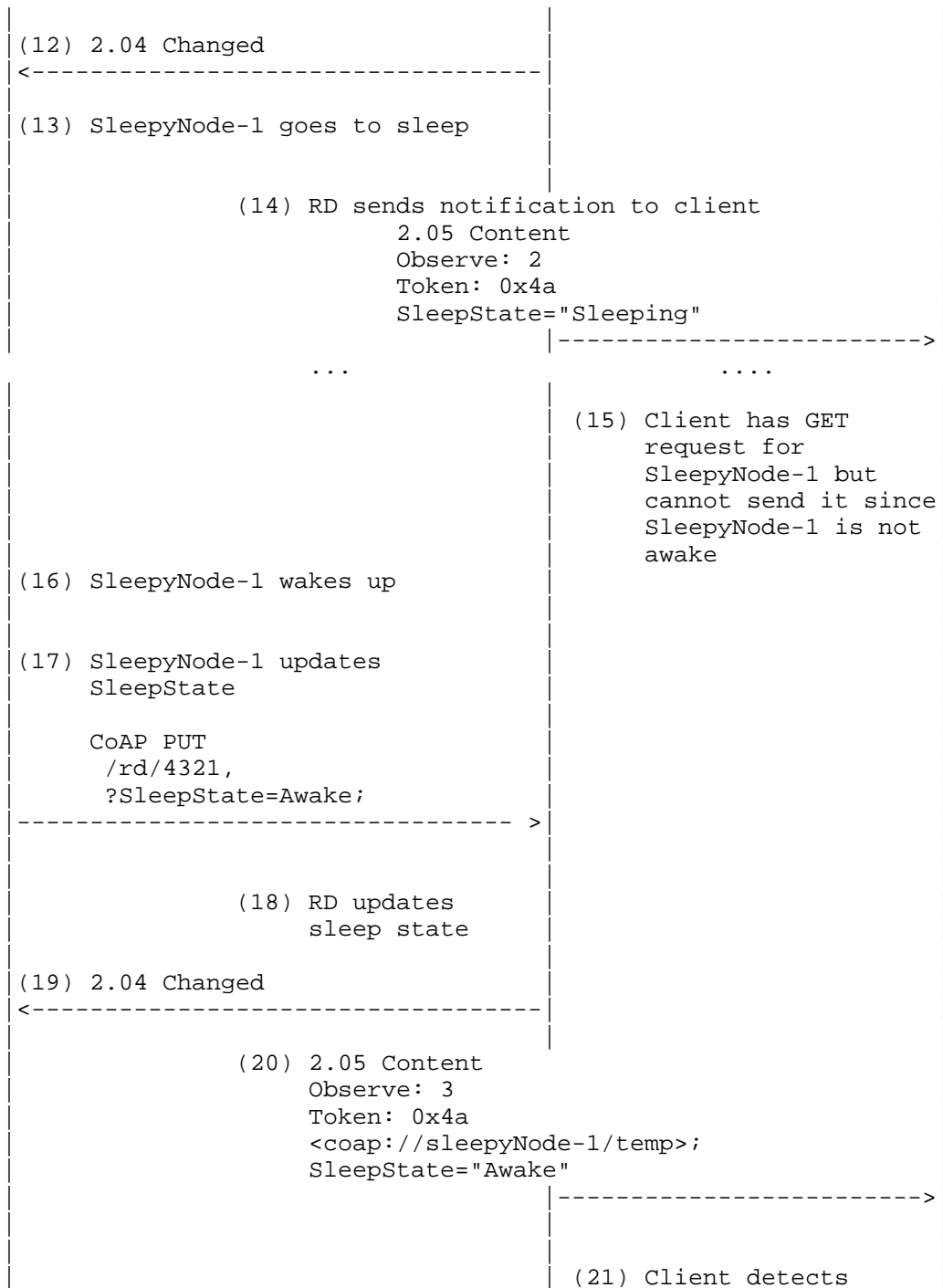
(20)-(21) RD notifies the client through previously established observe relationship.

(22)-(23) Client sends GET request for temperature sensors and successfully receives the content as SleepyNode-1 is awake.









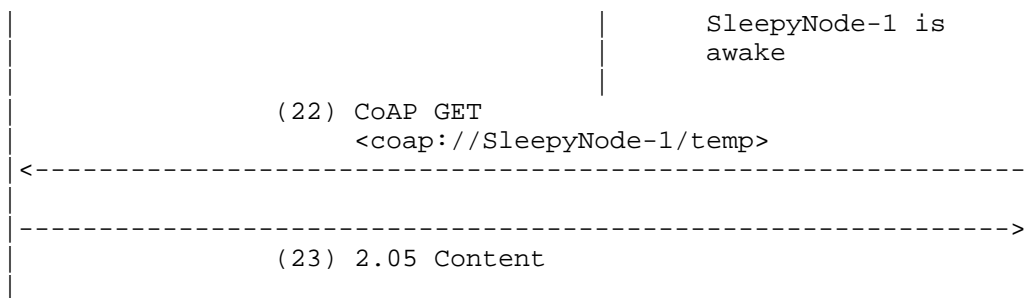


Figure 2: Asynchronous Resource Directory Based Sleep Tracking

### 3.4. RD Caching Proxy

It would be useful for an RD to be able to indicate which proxy performs caching for Sleepy CoAP nodes (see Section 2). This would be done through a new RD "CachingProxy" attribute for each device (similar to the attributes defined in Section 3.1):

- o An RD may be co-located with a proxy that performs caching for CoAP nodes. In this case, the RD automatically adds itself to each CachingProxy entry.
- o The sleepy node itself could suggest the CachingProxy if it is peered to a specific proxy.

This parameter would be added to the RD URI Template Variables defined in [I-D.ietf-core-resource-directory].

## 4. Performance Results

### 4.1. Prototype Setup

A simple prototype was implemented to validate certain aspects of the performance of the proposed CoAP sleepy node support protocol enhancement. The network for the prototype is shown in Figure 3.

Key aspects of the prototype are as follows:

- o There are two modes of operation: "Caching Only" and "Caching and Sleep Aware"
- o In "Caching Only" mode the Reverse Proxy will cache responses and service new requests according to the "Max-Age" parameter as defined in [I-D.ietf-core-coap].

- o In "Caching and Sleep Aware" mode the Reverse Proxy will also cache responses and service new requests according to "Max-Age". However, if the cache is empty (e.g. exceeded Max-Age) then there will be extra sleep aware functionality. Specifically, the proxy will also be aware of the "SleepDuration", "NextSleep" and "SleepState" sleepy node parameters as defined in Section 3.1. Based on these sleep parameters:
  - \* For servers that are asleep but expected to wake up soon: The proxy will immediately send a non-piggy backed ACK to the client. Then store-and-forward the request to the server when it wakes up, and then return the non-piggybacked response to the client thereafter.
  - \* For servers that are asleep but not expected to wake up soon: The proxy will immediately send a "5.03 Service Unavailable (and retry after)" to the client.
- o The key variables in the experiment are: (1) Number of clients; (2) Number of sleepy servers; (3) Delay between client requests; (4) MaxAge; (5) SleepDuration; (6) NextSleep; and (7) SleepState.
- o The target of the experiment will be to measure the amount of traffic over the network in the two modes of operation (for the same input client requests profile). It is hypothesized that the "Caching and Sleep Aware" mode will have the minimal amount of network traffic indicating that the Sleep Aware network will be the most efficient.

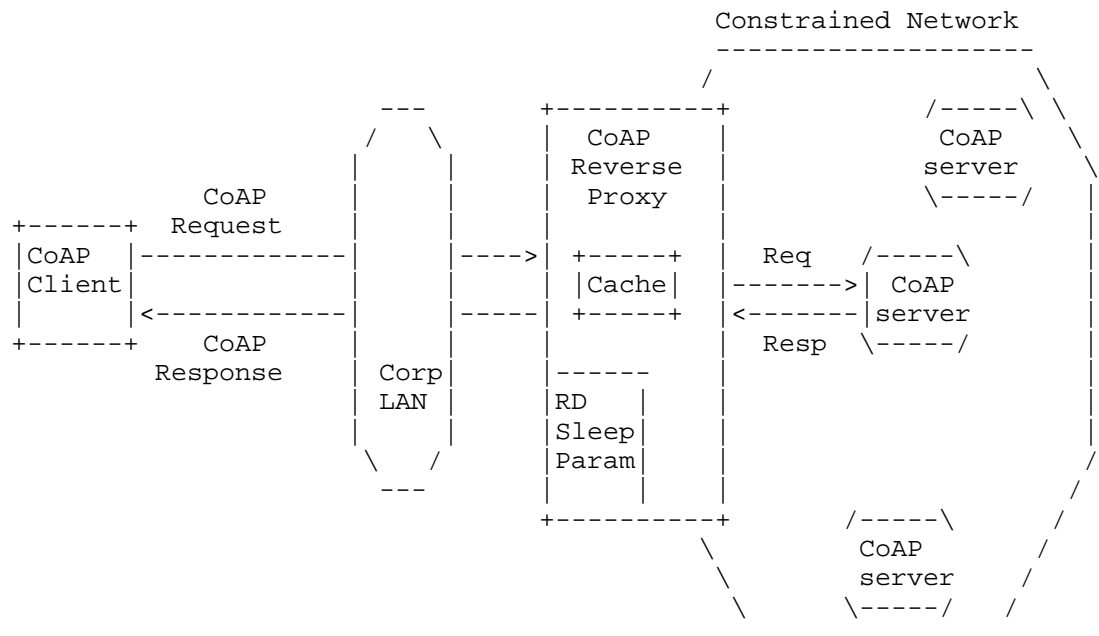


Figure 3: Prototype Configuration

#### 4.2. Initial Results

Baseline performance results of A CoAP system with Sleep Aware Proxy support described in Section 4.1 is given in [IETF86Results] (see pg. 65-95). Key conclusions were:

- o If only GETs are considered, a cache based system is efficient as long as the cache is valid (i.e before Max-Age).
- o However, since the cache often expires, a Sleep Aware Proxy further minimizes the number of GET transactions. Also, the Sleep Aware Proxy reduces number of transactions for other methods (e.g. PUTs) where a cache does not help.

#### 4.3. Comparison with Observe

A comparison of the performance of a CoAP system with Sleep Aware Proxy versus a CoAP system supporting Observe is given in [IETF87Results] (see pg. 150-182). Key conclusions were:

- o If only GETs are considered, an Observe based system is quite efficient.

- o However, Observe in combination with Sleep Aware Proxy further minimizes the number of GET transactions. Also, the Sleep Aware Proxy reduces number of transactions for other methods (e.g. PUTs) where Observe does not help.

## 5. Acknowledgements

Thanks to Carsten Bormann, Esko Dijk, Thomas Fossati, Salvatore Loreto, Dale Seed, and Zach Shelby for valuable discussions and feedback on this document.

## 6. IANA Considerations

This memo includes no request to IANA.

## 7. Security Considerations

TBD. (All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.)

## 8. References

### 8.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-11 (work in progress), October 2013.
- [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.

### 8.2. Informative References

- [I-D.ietf-core-resource-directory]  
Shelby, Z., Bormann, C., and S. Krco, "CoRE Resource Directory", draft-ietf-core-resource-directory-01 (work in progress), December 2013.

[I-D.rahman-core-sleepy-nodes-do-we-need]

Rahman, A., "Sleepy Devices: Do we need to Support them in CORE?", draft-rahman-core-sleepy-nodes-do-we-need-01 (work in progress), February 2014.

[IETF86Results]

Rahman, A., "IETF-86 Sleepy Node Performance Results", March 2013, <<http://www.ietf.org/proceedings/86/slides/slides-86-core-1.pdf>>.

[IETF87Results]

Rahman, A., "IETF-87 Sleepy Node Performance Results", July 2013, <<http://www.ietf.org/proceedings/87/slides/slides-87-core-0.pdf>>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

#### Author's Address

Akbar Rahman  
InterDigital Communications, LLC  
Montreal, Quebec H3A 3G4  
Canada

Phone: +1-514-585-0761  
Email: [akbar.rahman@interdigital.com](mailto:akbar.rahman@interdigital.com)

Core  
Internet-Draft  
Intended status: Standards Track  
Expires: January 11, 2013

B. Sarikaya  
Huawei USA  
Y. Ohba  
Toshiba  
R. Moskowitz  
Verizon Business Systems  
Z. Cao  
China Mobile  
R. Cragie  
Pacific Gas and Electric  
July 10, 2012

Security Bootstrapping Solution for Resource-Constrained Devices  
draft-sarikaya-core-sbootstrapping-05

Abstract

This document describes how to initially configure the network of resource constrained nodes securely, a.k.a., security bootstrapping. Bootstrapping architecture, communication channel and bootstrap security methods are described. System level objectives for security bootstrapping are stated. Bootstrapping solution is based on EAP-TLS authentication with the use of raw public keys as certificates.

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 January 11, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal

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



## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .  | 4  |
| 2. Bootstrapping Architecture . . . . .  | 4  |
| 2.1. Areas of Bootstrapping . . . . .  | 4  |
| 2.2. Architecture . . . . .  | 6  |
| 3. Bootstrap Security Method . . . . .   | 7  |
| 3.1. None . . . . .  | 7  |
| 3.2. Asymmetric with User Authentication, Followed by<br>Symmetric . . . . .               | 7  |
| 3.3. Asymmetric with Certificate Authority, Followed by<br>Symmetric . . . . .             | 7  |
| 3.4. Cryptographically Generated Address Based Address<br>Ownership Verification . . . . . | 7  |
| 4. Secure Bootstrapping System Level Objectives . . . . .                                  | 8  |
| 5. Secure Bootstrapping Solution using Raw Public Keys . . . . .                           | 9  |
| 6. Security Considerations . . . . .   | 10 |
| 7. IANA Considerations . . . . .   | 11 |
| 8. Contributors . . . . .  | 12 |
| 9. Acknowledgements . . . . .  | 12 |
| 10. References . . . . .   | 12 |
| 10.1. Normative References . . . . .   | 12 |
| 10.2. Informative References . . . . .   | 13 |
| Appendix A. Examples of Node Configuration . . . . .                                       | 15 |
| A.1. Smart Energy . . . . .  | 15 |
| A.1.1. Initial Meter Installation . . . . .  | 15 |
| A.1.2. Home Expansions . . . . .   | 15 |
| A.2. Consumer Products . . . . .   | 15 |
| A.2.1. Connecting DVD Remote to DVD Player . . . . .                                       | 16 |
| A.2.2. Adding a TV to a network with a DVD player and<br>remote . . . . .                  | 16 |
| A.2.3. Providing GPS Location Data . . . . .   | 16 |
| A.3. Commercial Building Automation . . . . .  | 16 |
| A.3.1. Light Installation . . . . .  | 16 |
| Appendix B. Example Exchanges . . . . .  | 16 |
| B.1. Smart Energy: Meter Manufacture . . . . .   | 16 |
| B.2. Smart Energy: Meter Installation . . . . .  | 16 |
| B.3. Smart Energy: Home Expansion . . . . .  | 16 |
| B.4. Consumer: Connecting DVD Remote to DVD Player . . . . .                               | 17 |
| B.5. Consumer: Adding a TV to a network with a DVD player<br>and remote . . . . .          | 18 |
| Appendix C. EAP, PANA, HIP-DEX and 802.1X . . . . .  | 20 |
| C.1. EAP Authentication Framework . . . . .  | 20 |
| C.2. PANA . . . . .  | 22 |
| C.3. HIP-DEX . . . . .   | 23 |
| C.4. 802.1X . . . . .  | 24 |
| Authors' Addresses . . . . .   | 26 |

## 1. Introduction

Bootstrapping is any processing required before the network can operate. Typically this will require a number of settings to be transferred between nodes at all layers. This could include anything from link-layer information (i.e., wireless channels, link-layer encryption keys) to application-layer information (i.e., network names, application encryption keys).

Bootstrapping is complete when settings have been securely transferred prior to normal operation in the network.

The bootstrapping problem is not specific to any MAC or PHY. This problem exists across any two nodes which have no previous knowledge of each other. In particular, this problem is complicated when the nodes are resource-constrained and may not have an advanced user interface. The IETF is instrumental in defining standards which will be used by The Internet of Things (IOT). Ensuring these standards can be used across nodes and networks requires some form of bootstrapping which the sensor nodes can use [ROMER04].

Existing standards will be used as much as possible in this document. The method proposed here should work across many different underlying layers. It could be used to allow two nodes on the same physical network to join at the physical layer, or allow two nodes on an incompatible physical network to join at the IPv6 layer.

The document continues in Section 2 on bootstrapping architecture, in Section 3 on allowable security methods in bootstrapping, in Section 4 on system level objectives of secure bootstrapping, and Section 5 on secure bootstrapping solution. Appendix A is on examples of node configuration, Appendix B is on samples of exchanges during bootstrapping and Appendix C is on the protocols used to carry EAP in link layer/IP layer including HIP-DEX.

## 2. Bootstrapping Architecture

### 2.1. Areas of Bootstrapping

In order to provide a flexible architecture, the bootstrapping method is split into five distinct areas and two distinct phases. The five areas are a 'user interface', 'bootstrap profile', 'security method', 'bootstrap protocol', and the 'communications channel'.

The phases are provisioning phase and bootstrapping phase. In the provisioning phase, statically configured parameters (e.g., certificates) needed for the bootstrapping phase is provisioned. In

the bootstrapping phase, dynamically configured information is set up using the statically configured information provided in the provisioning phase.

The user interface provides both user input and user output. Simple nodes may only have a push-button and LED, more complex nodes may have a graphical display and keyboard. The user interface (which could be implemented as network management software graphical user interface running at the remote end) provides interaction between the user and bootstrapping methods. The user interface would be used during bootstrapping as an OOB channel. It may also be used to specify bootstrapping policies.

The user interface provides the interaction between the user and the bootstrap protocol. The user interface will vary depending on the capabilities of the node. Examples might include a push-button and LED on simple nodes, to full-blown graphical user interfaces. Note that a 'bootstrapping tool' used to initially deploy a network is just a special user interface. This allows a very uniform protocol in deployment and use of networks.

User interface is out-of-scope and will not be further discussed.

Two nodes communicate through some channel. For our purposes this is split into the 'control channel' and 'data channel'. The control channel is used for the bootstrap protocol, and the data channel is used during normal network operation. A node may support multiple control or data channels. When the control and data channels are the same, the bootstrapping is done In Band (IB). When the control and data channels are different, the bootstrapping is performed Out Of Band (OOB). An 802.15.4 network for instance would use an 802.15.4 control channel for IB bootstrapping, but a control channel of perhaps IrDA or USB for OOB bootstrapping.

The 'bootstrap profile', i.e. statically configured parameters during the provisioning phase, defines what information should be exchanged during the process. A single node may run the protocol multiple times with different profiles. If the user wishes to associate a new lightswitch, the protocol is first run with the '802.15.4 Wireless Profile', through which it learns the channel and PAN-ID. The node then runs a 'Security Exchange Profile' to learn the needed encryption keys. Finally it runs a 'Lightswitch Association Profile' through which it learns which light to associate with.

An example of the 'bootstrap profile' attribute is the 'administrative domain name'. 'Bootstrap profiles' are required to be modified when the corresponding administrative domains are changed, a.k.a. recommissioning. In recommissioning, the domains are

administratively repartitioned and nodes are therefore recommissioned.

The 'security method' defines supported security methods for bootstrapping. The supported security methods will depend on the control channel and bootstrap profile. In one node if the control channel is secure, then a simple clear-text security method is supported. For example when a physical connection between two nodes is used, the control channel is considered secure. However when the control channel is not secure, this clear-text security method is not supported. The 'bootstrap profile' additionally defines allowed security methods. Higher security nodes may outlaw ever performing a clear-text exchange, even if the control channel is deemed secure.

The 'bootstrap protocol' defines the actual messages exchanged during bootstrapping. The messages are used to transfer between nodes data, node information, and network state. The selected security method runs on top of the control channel, such as EAP-GPSK etc.

## 2.2. Architecture

Security bootstrapping architecture is structured in a hierarchy of nodes going from the least resource constraint to the most resource constraint. At the top there is a root node. The root node is called Coordinator or Trust Center in Zigbee and 6LoWPAN Border Router (6LBR) in 6LoWPAN ND.

At the next level there are interior Routers. Routers are able to run a routing protocol between other routers and the root. Routers are called 6LoWPAN Routers (6BR) in 6LoWPAN ND.

At the lowest level there are the nodes. The nodes do not run a routing protocol. They can connect to the nearest router over a single radio link. The nodes are called End Devices in Zigbee and hosts in 6LoWPAN ND.

Routers first join the network as a node and go through security bootstrapping operations in order to create a Master Session Key (MSK). Next, routers execute routing protocol, e.g. [RFC6550] specific steps to create session keys with their neighbors and to establish upstream and downstream next hop parents.

At each node hierarchy level described above, there are lower-layer and higher-layer protocols to bootstrap their ciphering keys, where the lower-layer refers to layers below IP layer including IEEE 802.15.4 MAC layer and LoWPAN adaptation layer and the higher-layer refers to IP layer and the above. In general, required bootstrapping procedures depend on the bootstrapping protocols to use. Section

Section 5 describes the bootstrapping procedures where EAP (Extensible Authentication Protocol) [RFC3748] and other protocols are used as the bootstrapping protocols.

### 3. Bootstrap Security Method

The bootstrap security method defines allowable security methods. A node may choose to support or use a subset of these methods. This is NOT the security architecture used for the application, but only the security used during bootstrapping. Typically some high-security method is used to generate a shared secret, which then switches to simpler symmetric encryption to secure the actual bootstrapping channel. The techniques negotiated should take advantage of hardware resources available, such as hardware encryption accelerators on an end node.

#### 3.1. None

This is the simplest security method. No encryption or authentication is provided, messages are exchanged completely in clear-text. It is assumed some other layer provides security, such as a physical connection between devices.

#### 3.2. Asymmetric with User Authentication, Followed by Symmetric

A Diffie-Hellman style key exchange is used to generate a shared secret. The authentication will be provided by the user, by confirming cryptographic signatures between two devices. With the shared secret generated through the DH, some symmetric encryption is used to secure the actual bootstrapping channel.

#### 3.3. Asymmetric with Certificate Authority, Followed by Symmetric

Public key exchanges are used (aka: DH again), but with a Certificate Authority. Once a shared secret exists, symmetric encryption is used to secure the actual bootstrapping channel.

#### 3.4. Cryptographically Generated Address Based Address Ownership Verification

A node may generate the global unique address using different techniques other than the stateless address autoconfiguration. For example, Cryptographically Generated Addresses (CGA) [RFC3972] is a type of global unique address that can be used to verify the address ownership. When the node uses CGA, it MUST execute SeND protocol [RFC3971]. In a 6LOWPAN network, a modified 6LOWPAN ND Protocol [I-D.ietf-6lowpan-nd] must be executed between the node and the edge

router.

#### 4. Secure Bootstrapping System Level Objectives

Authentication/ reauthentication: nodes joining the network MUST at the first place authenticate to the trust center. In order to achieve secure multi-hop routing, the node MUST authenticate to its upstream and downstream neighbors. A bootstrapping solution MUST support re-authentication of resource-constrained devices and re-keying of dynamically generated keys.

Data Confidentiality: the data communication between two endpoints MAY be encrypted using the derived key, avoiding being eavesdropped by a non-trusted third part.

Data Integrity: the data communication between two endpoints MUST NOT be altered by some intermediate nodes. The nodes should be able to detect the non-integral data.

Keys and key freshness: the keys used for data communication MUST have a lifetime, in order to keep their freshness. A bootstrapping solution MUST support both symmetric and asymmetric key authentication. If distribution of a key to be used for a resource-constrained device is required, a bootstrapping solution MUST support secure key distribution to prevent the key from eavesdropping, alternation and replay attacks.

Multi domain support: A bootstrapping solution MUST be able to allow resource-constrained devices that may be subscribed to different administrative domains to be connected to the same access network at the same time.

Multi domain support: A bootstrapping solution MUST be able to allow resource-constrained devices to be recommissioned. Recommissioning a device is defined to be (1) an resource-constrained device is administratively switched to a different domain, or (2) acting a new role with a different function set, or (3) both administrative domain and function set are modified.

Identities: A bootstrapping solution MUST be able to allow a resource-constrained device to use various types of identities used for authentication, including device identities, user identities or combinations of different types of identities. Also a bootstrapping solution MUST be able to allow a resource-constrained device to change its identities used for authentication over time.

Authentication infrastructure: A bootstrapping solution MUST be able

to operate with or without an authentication infrastructure.

## 5. Secure Bootstrapping Solution using Raw Public Keys

When a new resource-constrained device is deployed, it configures its global unique IPv6 address first. This is done by 6LoWPAN Neighbor Discovery (6LoWPAN-ND)'s Router Solicitation/Router Advertisement message exchange [I-D.ietf-6lowpan-nd]. The newly generated IPv6 address can not be used until the joining device is authenticated and securely joins the network. After the authentication, the joining device receives the current group key of the network, so that the IPv6 registration and further communication can be protected by the link layer ciphering e.g. 802.15.4, then it can start using its global unique IPv6 address for communication.

For authentication, Extensible Authentication Protocol (EAP) MUST be used. EAP authentication framework is explained in Appendix C.1.

The EAP method EAP-TLS [RFC5216] can be used for the resource-constrained device authentication. Instead of X.509 certificates, raw public key of the device MUST be used. EAP-TLS is executed between the joining device and the AAA server which acts as the Authentication Server (AS). After a successful authentication, the device and the AAA server establish a Master Session Key (MSK), and then the AAA server exports the MSK to the authenticator. Upon receipt of the MSK, the authenticator distributes the group key to the joining device within the authentication success message. The group key is encrypted by a Key Encryption Key derived from the MSK.

The resource-constrained device initiates the EAP authentication process by sending a message of initiation to the authenticator, i.e. the root node or 6LBR. The root node requests the identity from the device by sending an EAP-Request/Identity packet. The device replies with an EAP-Response/Identity containing the device's ID. The identity information includes the device's network access ID (NAI). When the root node receives NAI of the device, it sends the identity information to the AS.

The AS starts the EAP-TLS authentication process by sending a EAP-TLS/Start packet which is an EAP-Request packet with EAP-Type=EAP-TLS to the device. The device generates a client random number and responds with an EAP-Response/TLS-Client-Hello message which contains the TLS version, a client random number, a set of cipher suites. Only one cipher suite MUST be offered in Client-Hello message with RC4-SHA1. EAP-Response packet MUST have the EAP- Type value set at EAP-TLS.

The device MUST add an extension of type `cert_type` defined in [I-D.ietf-tls-oob-pubkey] to Client-Hello message. This type MUST be set to `RawPublicKey`.

Upon receipt of Client Hello, if the AS supports raw public key extension, it generates a server random number, a new session ID, and includes only the `SubjectPublicKeyInfo` part of the certificate, rather than the whole certificate in the Certificate message and then sends them to the device with an EAP-Request/TLS-Server-Hello message.

With the client and server random number, the device generates a `pre_master_secret`, then sends it in Client-Key-Exchange field of EAP-Response/TLS-Client-Finished message to the AS.

The AS derives the Master Session Key (MSK) and replies with EAP-Request/TLS-Server-Finished message. The SE device also derives the MSK after receiving the Server Finished and acknowledges with EAP-Response/EAP-TLS message.

The AS then exports the MSK to the authenticator in RADIUS Access-Accept message, the authenticator subsequently sends the EAP-Success message to the SE device. The AS MUST send the group key in this message and the EAP-TLS ends.

When a device is not a direct neighbor of the authenticator, its parent node MUST act as relay. Different EAP encapsulation protocols have different mechanisms for the relay function, for details, see Appendix C.

## 6. Security Considerations

When security bootstrapping resource constraint nodes is undertaken, several attacks are possible and security bootstrapping methods described in this document do not protect the nodes against such attacks. These attacks are similar to the ones described in [RFC3971] and mainly stem from unsecured link layer. Link layer must be secured on each node before the node can begin security bootstrapping.

If a bootstrapping protocol does not rely on a pre-shared key for peer authentication, it must rely on an online or offline third-party (e.g., an authentication server, a key distribution center in Kerberos, a certification authority in PKI, a private key generator in ID-based cryptography and so on) to prevent man-in-the-middle attacks during peer authentication. Depending on use cases, a resource-constrained device may not always have access to an online



third-party for peer authentication.

Depending on use cases, a bootstrapping protocol may deal with authorization separately from authentication in terms of timing and signaling path. For example, two resource-constrained devices A and B may perform mutual authentication using authentication credentials provided by an offline third-party X whereas resource-constrained device A obtains authorization for running a particular application with resource-constrained device B from an online third-party Y before or after the authentication. In some use cases, authentication and authorization are tightly coupled, e.g., successful authentication also means successful authorization. A bootstrapping protocol supports various types of authentication and authorization or different bootstrapping protocols may be used for different types of authentication and authorization.

If authorization information includes cryptographic keys, a special care must be taken for dealing with the keys, e.g., guidelines for AAA-based key management are described in [RFC4962]. A recommissioning use case may require revocation and re-installation of authentication credentials (i.e., a certificate or a shared secret and identity information, etc.) to a large number of resource-constrained devices that are already deployed. Re-installation of authentication credentials must be as secure as the initial installation regardless of whether the re-installation is done manually or automatically.

If resource-constrained devices use a multicast group key for peer authentication or message authentication or encryption, the group key must be securely distributed to the current members of the group for both initial key distribution and key update. Protocols designed for group key management such as GSAKMP [RFC4535], GDOI [RFC3547] and MIKEY [RFC3830] may be used for group key distribution. Alternatively, key wrap attributes for securely encapsulating group key may be defined in network access authentication protocols such as PANA [RFC5191] and EAP-TTLSv0 [RFC5281]. Those protocols use an end-to-end, point-to-point communication channel with a pair-wise security association between a key distribution center and each key recipient. Further considerations may be needed for more efficient group key management to support a large number of resource-constrained devices.

## 7. IANA Considerations

This memo includes no request to IANA.

## 8. Contributors

The people listed below have made significant text contributions to this document.

Colin O'Flynn

colin.oflynn@atmel.com

## 9. Acknowledgements

Special thanks also to Rene Struik, Carsten Borman, Gary Yang, Alper Yegin and Tero Kivinen for their comments that helped us improve the writing. Discussions with Hannes Tschofenig have been very inspiring.

## 10. References

### 10.1. Normative References

- [802.15.4] IEEE Std 802.15.4-2006, "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)", September 2006.
- [I-D.ietf-tls-oob-pubkey] Wouters, P., Gilmore, J., Weiler, S., Kivinen, T., and H. Tschofenig, "TLS Out-of-Band Public Key Validation", draft-ietf-tls-oob-pubkey-03 (work in progress), April 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.
- [RFC5191] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)", RFC 5191, May 2008.

- [RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, March 2008.
- [RFC5548] Dohler, M., Watteyne, T., Winter, T., and D. Barthel, "Routing Requirements for Urban Low-Power and Lossy Networks", RFC 5548, May 2009.
- [RFC5673] Pister, K., Thubert, P., Dwars, S., and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks", RFC 5673, October 2009.

## 10.2. Informative References

- [802.1x] IEEE Std 802.1X-2010, "IEEE 802.1X Port-Based Network Access Control", February 2010.
- [C1222] American National Standard, "Protocol Specification For Interfacing to Data Communication Networks", ANSI C12.22-2008, 2008.
- [I-D.ietf-6lowpan-nd] Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor Discovery Optimization for Low Power and Lossy Networks (6LoWPAN)", draft-ietf-6lowpan-nd-18 (work in progress), October 2011.
- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.moskowitz-hip-rg-dex] Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-06 (work in progress), May 2012.
- [I-D.ohba-pana-relay] Duffy, P., Chakrabarti, S., Cragie, R., Ohba, Y., and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA) Relay Element", draft-ohba-pana-relay-03 (work in progress), February 2011.
- [NISTIR7628VOL1] The Smart Grid Interoperability Panel - Cyber Security Working Group, "Guidelines for Smart Grid Cyber Security: Vol. 1, Smart Grid Cyber Security Strategy, Architecture, and High-Level Requirements", NISTIR 7628, vol. 1, 2010.

- [RFC3547] Baugher, M., Weis, B., Hardjono, T., and H. Harney, "The Group Domain of Interpretation", RFC 3547, July 2003.
- [RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.
- [RFC3971] Arkko, J., Kempf, J., Zill, B., and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005.
- [RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.
- [RFC4423] Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture", RFC 4423, May 2006.
- [RFC4535] Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, June 2006.
- [RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.
- [RFC5204] Laganier, J. and L. Eggert, "Host Identity Protocol (HIP) Rendezvous Extension", RFC 5204, April 2008.
- [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.
- [RFC5281] Funk, P. and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.
- [RFC5295] Salowey, J., Dondeti, L., Narayanan, V., and M. Nakhjiri, "Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK)", RFC 5295, August 2008.
- [RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen,

"Internet Key Exchange Protocol Version 2 (IKEv2)",  
RFC 5996, September 2010.

- [RFC6550] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R.,  
Levis, P., Pister, K., Struik, R., Vasseur, JP., and R.  
Alexander, "RPL: IPv6 Routing Protocol for Low-Power and  
Lossy Networks", RFC 6550, March 2012.
- [ROMER04] Romer, K. and F. Mattern, "The design space of wireless  
sensor networks", IEEE Wireless Communications, vol. 11,  
no. 6, pp. 54-61, December 2004.
- [SE2.0] ZigBee Alliance, "Smart Energy Profile 2.0 Technical  
Requirements Document", April 2010.

## Appendix A. Examples of Node Configuration

Before any detail on methods is explored, the following section will provide various examples this document could cover. Exact requirements will be brought forward in subsequent sections. For the reader's general understanding this section is placed to give an idea of an acceptable usage scenario.

### A.1. Smart Energy

#### A.1.1. Initial Meter Installation

The meter is initially loaded with code and network keys through a physical interface at the factory. The meter is installed at a customers home, and configured by the installer through the backbone link (via GSM modem, etc). Both operations can be performed through methods defined herein.

#### A.1.2. Home Expansions

The user wishes to join a thermostat onto the network. They press a button on the thermostat, which enters join mode. They press a button on the smart meter, which allows nodes to join the network. The devices both have displays, so they display a certain number which the user verifies match on both devices. The thermostat has now securely joined the network.

### A.2. Consumer Products

#### A.2.1. Connecting DVD Remote to DVD Player

The user pushes a join button on the DVD remote and DVD player. The devices find each other, and blink in unison to indicate to the user which two devices will join. The user presses the button to confirm this, and the two devices are now joined together.

#### A.2.2. Adding a TV to a network with a DVD player and remote

The user then presses the join button on the DVD player and TV. The devices again find each other and blink in unison, with the addition that the remote control also blinks to indicate it is present in the network.

#### A.2.3. Providing GPS Location Data

A user has a simple GPS receiver (that has no user interface) they wish to broadcast location data with. The user switches on their camera, and enters a PIN from the base of the GPS. The user can now view GPS information such as satellite health from their camera. In addition photos are automatically tagged with location information.

### A.3. Commercial Building Automation

#### A.3.1. Light Installation

The electrician installs the light fixture. Each light has a barcode printed on it. They use a handheld barcode scanner tool, which acts as the commissioning tool. When they scan a barcode with the tool, the tool asks the electrician to enter some additional information such as light fixture location. The tool securely registers the light fixture on the network, along with setting parameters inside the light fixture.

## Appendix B. Example Exchanges

The following details how the protocol handles certain conditions and situations through examples. Note that each example is a more detailed description of the examples in Appendix A.

#### B.1. Smart Energy: Meter Manufacture

#### B.2. Smart Energy: Meter Installation

#### B.3. Smart Energy: Home Expansion

## B.4. Consumer: Connecting DVD Remote to DVD Player

Supported User Interface Profiles

| Profile      | DVD Player | Remote Control |
|--------------|------------|----------------|
| none         | Y          | Y              |
| simple       | Y          | Y              |
| numerical    | Y          | N              |
| alphanumeric | Y          | N              |
| Graphical    | Y          | N              |

Supported Bootstrap Transport Layers

| Layer    | DVD Player | Remote Control |
|----------|------------|----------------|
| Physical | Y          | Y              |
| 802.15.4 | Y          | Y              |
| IrDA     | Y          | N              |

Supported Security Methods

| Method           | DVD Player | Remote Control |
|------------------|------------|----------------|
| None             | Y          | Y              |
| EAP              | Y          | N              |
| Asymmetric, User | Y          | Y              |
| Asymmetric, CA   | Y          | N              |

The DVD player and remote control have a number of ways in which they could be joined together. The remote control does not have any unique identifier printed on it, thus no pre-shared key can be identified. This leaves either an unsecure joining method, or some asymmetric security method.

The remote control has a button on it for 'join', as does the DVD player. The user pushes the button on the DVD player, and then pushes the button on the remote control. Based on the UI profile, this causes the following to occur:

- o DVD Player scans for existing network in advertise mode. Finding none, it starts a new network and that network enters advertise

mode.

- o The DVD remote scans for a network, and then finds the DVD player's network.
- o The devices generate a shared secret (ie: Diffie-Hellman), and both blink their LED in a unique pattern based on this shared secret.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The DVD player displays 'JOIN OK' on it's LCD panel.

#### B.5. Consumer: Adding a TV to a network with a DVD player and remote

This network will have three devices: a TV, a DVD Player, and a Remote Control. The user will run the bootstrap protocol between the TV and Remote Control in this example, although it could also be run between the TV and DVD player.

Supported User Interface Profiles

| Profile      | TV | Remote Control |
|--------------|----|----------------|
| none         | Y  | Y              |
| simple       | Y  | Y              |
| numerical    | Y  | N              |
| alphanumeric | Y  | N              |
| Graphical    | Y  | N              |

Supported Bootstrap Transport Layers

| Layer    | TV | Remote Control |
|----------|----|----------------|
| Physical | Y  | Y              |
| 802.15.4 | Y  | Y              |
| IrDA     | Y  | N              |



## Supported Security Methods

| Method           | TV | Remote Control |
|------------------|----|----------------|
| None             | Y  | Y              |
| EAP-GPSK         | Y  | N              |
| Asymmetric, User | Y  | Y              |
| Asymmetric, CA   | Y  | N              |

The TV and remote control have a number of ways in which they could be joined together. The remote control does not have any unique identifier printed on it, thus no pre-shared key can be identified. This leaves either an unsecure joining method, or some asymmetric security method.

The remote control has a button on it for 'join', as does the TV. In this example two sequence will be considered: where the TV button is pressed first, and where the remote control button is pressed first.

If the TV join button is pressed first:

- o TV scans for existing networks in advertise mode. Finding none, it starts a new network and that network enters advertise mode.
- o The remote scans for a network, and then finds the TV's network.
- o The remote informs the TV it is on an existing network, and thus will require the TV to join this network.
- o The devices generate a shared secret, and both blink their LED in a unique pattern.
- o The DVD player in addition blinks, so the user is informed that if they confirm the join action the resulting network will have all three devices in it.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The TV displays 'JOIN OK' onscreen, along with any information about the network it just joined.

If the remote control join button is pressed first:

- o Remote control scans for existing networks in advertise mode. Finding none, it advertises it's network.

- o The TV scans for a network, and then finds the remote control's network.
- o The devices generate a shared secret, and both blink their LED in a unique pattern.
- o The DVD player in addition blinks, so the user is informed that if they confirm the join action the resulting network will have all three devices in it.
- o The user confirms both devices are blinking the same pattern, as both LEDs are blinking in unison.
- o The TV displays 'JOIN OK' onscreen, along with any information about the network it just joined.

## Appendix C. EAP, PANA, HIP-DEX and 802.1X

### C.1. EAP Authentication Framework

EAP is an authentication protocol that supports a number of authentication algorithms called EAP methods [RFC5247]. EAP messages can be transported in different layers: using 802.1X, EAP messages are carried in Layer 2 and using PANA in IP or Layer 3 between EAP peer and the authenticator. EAP messages between the authenticator and authentication server are carried using AAA protocols (RADIUS or Diameter). The associated AAA server address of each resource-constrained device is assigned by the domain administrator. In the recommissioning case, another AAA server is reassigned to devices by the domain administrator if the device is switched to another domain.

At the end of a successful EAP method execution a master session key (MSK) is generated at both the EAP peer and EAP server. Authenticator receives MSK from EAP server at the end of EAP method execution using key transport. MSK is used in deriving a session key between the node and the authenticator using a protocol called secure association protocol (SAP). Derivation of the session key terminates bootstrapping of a node.

Additional keying material derived between EAP client and server that is exported by the EAP method is called Extended Master Session Key (EMSK). EMSK is not used in session key derivation but it could be used for the needs of other applications in higher layer protocols.

In the architecture introduced in Section 2.2 the node or router is the peer and the root is the authenticator. When the supplicant and authenticator are one hop away the authenticator can be reached

directly. However, this is rarely the case. In other cases the authenticator authenticates neighboring supplicants first. The router nodes that are authenticated become relay authenticators in the next phase and they relay authentication messages from the supplicants to the authenticator and vice versa. This continues until all nodes are authenticated.

EAP is a lock-step protocol, i.e. it executes in pairs of EAP-Request messages sent by the server and EAP-Response messages sent by the peer. At the end, the server indicates the status of authentication, usually by EAP-Success message which also carries the MSK. The first EAP-Request/Response pair is used for the server to request the identity and the peer to provide it. In the other pairs of EAP exchanges EAP method is executed.

Several EAP methods have been standardized each for different purposes. To authenticate devices with certificates, EAP Transport Layer Security (TLS) v1.2 specified in [RFC5216] which supports certificate-based mutual authentication is used.

Zigbee Alliance's Smart Energy Profile 2.0 Application Protocol Specification [SE2.0] mandates each device to be factory programmed with a certificate. The certificate is bound to a unique network ID, e.g. the device's MAC address or EUI-64 address. During EAP-Identity exchange the EAP peer provides its EUI-64 address as an identity to EAP server. This enables the server to validate the device certificate.

There are three bootstrapping scenarios using EAP.

1. Use of EAP for bootstrapping link-layer security.

In this case, EAP is used for network access authentication to bootstrap link-layer ciphering. Security for higher-layer (i.e., IP layer and above) protocols is bootstrapped from an IB or OOB mechanism other than EAP.

2. Use of EAP for bootstrapping higher-layer security.

In this case, EAP is used as an OOB mechanism for higher-layer authentication to bootstrap ciphering keys for one or more higher-layer protocols independently of network access authentication. When bootstrapping Constrained Application Protocol (CoAP) security with DTLS protection, a PSK (Pre-Shared Key) credential in the combined usage of DTLS (Datagram Transport Layer Security) [RFC4347] and PSK mode of TLS [RFC4279] is derived from EAP key material and DTLS ciphering keys are generated as a result of a successful DTLS handshake. Similarly,

when bootstrapping CoAP security with IPsec ESP protection, a PSK credential of IKEv2 [RFC5996] is derived from EAP key material and IPsec ESP ciphering keys are generated as a result of a successful IKEv2 handshake.

The ability to bootstrap multiple higher-layer protocols from a single execution of PANA authentication is important to save the computational resources for resource-constrained devices especially where public-key based authentication is used.

3. Use of EAP for bootstrapping both link-layer and higher-layer security.

This case is the combination of the other two cases, and the most optimized way for bootstrapping resource-constrained devices. This case is only applicable where both the network access authentication and the higher-layer authentication use the same authentication server with the same authentication credentials.

The second and third scenarios are generally referred to as Single Sign-On in Section 4.2.2.2 of [NISTIR7628VOL1], where the root keys for higher-layer protocols can be derived from EAP EMSK (Extended Master Session Key) as an USRK (Usage-Specific Root Key) [RFC5295].

## C.2. PANA

PANA (Protocol for carrying Authentication for Network Access) [RFC5191] defines an EAP transport over UDP where a PANA Client (PaC) is an EAP peer and a PANA Authentication Agent (PAA) is an EAP authenticator.

PANA can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

Multi domain operation is intrinsically supported due to the use of EAP and AAA.

Even though PANA architecture consisting of PaC, PAA and AAA Server is generic enough to be used in security bootstrapping, the architecture introduced in Section 2.2 requires a new element called PANA Relay Element (PRE). PRE is needed to enable PANA messaging between a PaC and PAA because the two nodes cannot reach each other by means of regular IP routing since only a link-local IPv6 address can be used by PaC prior to the completion of a successful authentication.

PRE which is one hop away from PaC receives PANA messages and relays the message contents (payload) by encapsulating it in a message

parameter called Attribute Value Pair (AVP). PRE also needs to send header contents such as PaC's IP address and UDP port number in a different AVP. PRE has IP routing established with PAA which could be several hops away. PAA sends its reply messages in which the payload is encapsulated in an AVP. It also adds the AVP containing PaC's IP address and UDP port number. PRE creates a link local PDU using these AVPs and sends it to PaC [I-D.ohba-pana-relay].

The requirements for the use of PANA as a bootstrapping protocol can be stated as follows:

- o A new entity called PANA Relay Element may be added to the PANA architecture. Behaviour of PANA Relay Element needs to be defined. New AVPs needed for PANA Relay Element operation for properly relaying messages from the client to the authenticator and vice versa are required to be specified.
- o An extension to PANA to securely distribute keys from the PANA Authentication Agent to the PANA Client using AES Key Wrap with Padding algorithm needs to be defined. This is needed in order to use PANA for multicast group key distribution.

### C.3. HIP-DEX

[RFC4423] introduces the Host Identity Protocol (HIP) where the Host Identity (HI) is a Cryptographic key (RSA, DSA, or ECC). A 128-bit length Host Identity Tag (HIT) is derived from the HI (hashed) and functions as an IPv6 address (/128 prefix) for applications. A four-packet Peer-to-Peer Host Identity Protocol Base EXchange (HIP BEX) establishes a security association (SA, similar to IKE), indexed by the HITs, but independent of the IP address. So HIP can be considered as a shim layer between the transport(TCP/UDP) and IP, providing authentication, data confidentiality, mobility in one basket.

As with IKE, HIP is typically used as a Key Management Protocol (KMP) for ESP. However, HIP is independent of IP and ESP and can be used for a KMP for any secure communication protocol at any level. Thus HIP can provide keying material for the MAC, IP, and Transport layers. HIP can be run directly over the MAC in an Ethernet Frame or within an Information Element in a MAC control frame. HIP can be run over UDP, traversing firewalls, and push keys over to Transport security protocols like SRTP or (D)TLS. Further, HIP could start on the MAC, then be enveloped over UDP after the first link.

The HIP-BEX involves many crypto primitives that are difficult to run on constrained nodes. HIP Diet Exchange (HIP-DEX) [I-D.moskowitz-hip-rg-dex] is a way to make HIP lightweight.

Basically, HIP-DEX a variant of the HIP-BEX specifically designed to use as few crypto primitives as possible yet still provide the same class of security features as HIP-BEX.

HIP-DEX can be used for mutual authentication between two endpoints. After mutual authentication, the two endpoints establish a shared secret, which is fresh and fed into the encryption algorithm for data confidentiality. So HIP-DEX can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

When a node wants to authenticate to the network using HIP and Diet-HIP, it should be able to complete the HIP-BEX or HIP-DEX with the trust anchor or some delegate. In HIP, it does not matter how many domains, and nodes can authenticate each other as long as they have the secret.

In the architecture introduced in Section 2.2 the node and router could be the HIP end-points. Or the router could be a HIP Rendezvous Server, with the node registering to the rendezvous server (RVS) [RFC5204] to be reachable by other nodes. Typically the initial interaction will have the node be the HIP DEX Initiator with the router being the Responder. Using the RVS function on a Router any node could be the Initiator with any other Responder node. As long as there is IP connectivity between the Initiator and Responder, they can be multiple hops away from each other. Alternatively if the first hop from the Initiator has knowledge of the IP address of the Responder, it can act as a MAC/IP gateway for HIP.

An important requirement for the HIP-DEX to work in the architecture, the initiator should be able to get the IP address of the responder or have a gateway function as a forwarder. The IP address can be obtained from a 3rd party source like DNS or Distributed Hash Table (DHT), from local configuration, or from an RVS.

#### C.4. 802.1X

IEEE 802.1X defines how EAP packets can be transported over in Layer 2, i.e. Ethernet frames [802.1x] by encapsulating EAP packets into EAP Over Lan (EAPOL) frames between EAP peer, called supplicant and the authenticator. EAPOL can also be used in 802.11 wireless links.

To enable IEEE 802.15.4 devices to use EAP authentication, EAP packets encapsulated in EAPOL frames can be carried as payload in 802.15.4 data frames [802.15.4]. EAPOL is well defined and widely used and it lends itself to be easily carried in 802.15.4 data frames. For this, Frame Type subfield of the Frame Control Field of IEEE 802.15.4 MAC header needs to be set to a special value to

indicate the type of the payload, i.e. 802.1X encapsulated EAP packets. EAPOL packets are encoded following common EAPOL PDU structure defined in [802.1x] into the data payload field of 802.15.4 data frames.

Authentication proceeds as follows: authenticator authenticates the supplicants that are on the next hop first. This enables a secure link between the authenticator and these first-hop nodes. The architecture introduced in Section 2.2 requires a new entity called Relay Authenticator. First-hop nodes or router become Relay Authenticators in the next phase of authentication. Relay Authenticators tunnel EAPOL frames to the authenticator in the secure link established. This way all the supplicants are gradually authenticated.

After the keys are established from a successful EAP method (such as PSK mode of TLS), the node runs neighbor discovery protocol to get an IPv6 address assigned [I-D.ietf-6lowpan-nd]. Data transfer can be secured using DTLS or IPSec. Keys derived from EAP TLS are used in either generating DTLS ciphering keys after a successful DTLS handshake or IPSec ESP ciphering keys after a successful IKEv2 handshake.

802.1X can achieve the authentication, key freshness and data confidentiality objectives of security bootstrapping.

Multi domain operation is intrinsically supported due to the use of EAP and AAA. In order to support a device recommissioning case whereby the device's administrative domain is modified, after a new AAA server address assigned and a successful 802.1X method execution, the old set of device 'name and password' MUST be overwritten into the device by a new set of 'name and password' that are assigned by the domain administrator. The device MUST be rebooted to execute EAP method again for authentication and a new master session key MUST be generated.

It should be noted that currently 802.15.4 does not allow multiplexing multiple protocols on the same link like ethernet does. However this might change in the future.

The requirements for the use of 802.1X defined EAPOL as a bootstrapping protocol can be stated as follows:

- o A special value in the Frame Type subfield of the Frame Control Field of IEEE 802.15.4 MAC header to indicate the type of the payload,

- o Link Layer Multicast Group addresses for 802.15.4 corresponding to EAPOL Group Address Assignments defined in Table 11.1 of [802.1x], especially to be used in EAPOL-Start packet.
- o Which MAC frames of beacon, data, acknowledgment and MAC command as defined in [802.15.4] with what security levels are mapped to controlled port, which MAC frames with what security levels are mapped to uncontrolled port and which MAC frames are never mapped to any of controlled/uncontrolled port (i.e., the payload of those frames are used by the MAC-layer itself and never used by upper layers).
- o A new entity called Relay Authenticator may be added to the 802.1x architecture. Behaviour of Relay Authenticator needs to be defined.

#### Authors' Addresses

Behcet Sarikaya  
Huawei USA  
5340 Legacy Dr.  
Plano, TX 75024

Email: sarikaya@ieee.org

Yoshihiro Ohba  
Toshiba  
Tokyo, Japan

Email: yoshihiro.ohba@toshiba.co.jp

Robert Moskowitz  
Verizon Business Systems  
15210 Sutherland  
Oak Park, MI 48237

Email: rgm@labs.htt-consult.com

Zhen Cao  
China Mobile  
Beijing, China

Email: caozhen@chinamobile.com



Robert Cragie  
Pacific Gas and Electric  
89 Greenfield Crescent  
Wakefield, UK WF4 4WA

Email: robert.cragie@gridmerge.com



CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: September 17, 2013

Z. Shelby  
Sensinode  
M.V. Vial  
Schneider-Electric  
March 16, 2013

CoRE Interfaces  
draft-shelby-core-interfaces-05

Abstract

This document defines well-known REST interface descriptions for Batch, Sensor, Parameter and Actuator types for use in constrained web servers using the CoRE Link Format. A short reference is provided for each type that can be efficiently included in the interface description attribute of the CoRE Link Format. These descriptions are intended to be for general use in resource designs or for inclusion in more specific interface profiles. In addition, this document defines the concepts of Function Set and Binding. The former is the basis element to create RESTful profiles and the latter helps the configuration of links between resources located on one or more endpoints.

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 17, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .                      | 2  |
| 2. Terminology . . . . .                       | 3  |
| 3. Function Set . . . . .                      | 4  |
| 3.1. Defining a Function Set . . . . .         | 4  |
| 3.1.1. Path template . . . . .                 | 4  |
| 3.1.2. Resource Type . . . . .                 | 5  |
| 3.1.3. Interface Description . . . . .         | 5  |
| 3.1.4. Data type . . . . .                     | 5  |
| 3.2. Discovery . . . . .                       | 6  |
| 3.3. Versioning . . . . .                      | 6  |
| 4. Bindings . . . . .                          | 6  |
| 4.1. Format . . . . .                          | 7  |
| 4.2. Binding methods . . . . .                 | 8  |
| 4.3. Binding table . . . . .                   | 9  |
| 5. Interface Descriptions . . . . .            | 9  |
| 5.1. Link List . . . . .                       | 10 |
| 5.2. Batch . . . . .                           | 11 |
| 5.3. Linked Batch . . . . .                    | 11 |
| 5.4. Sensor . . . . .                          | 13 |
| 5.5. Parameter . . . . .                       | 13 |
| 5.6. Read-only Parameter . . . . .             | 14 |
| 5.7. Actuator . . . . .                        | 14 |
| 5.8. Binding . . . . .                         | 15 |
| 5.9. Resource Observation Attributes . . . . . | 15 |
| 5.10. Future Interfaces . . . . .              | 18 |
| 5.11. WADL Description . . . . .               | 18 |
| 6. Security Considerations . . . . .           | 22 |
| 7. IANA Considerations . . . . .               | 22 |
| 8. Acknowledgments . . . . .                   | 22 |
| 9. Changelog . . . . .                         | 22 |
| 10. References . . . . .                       | 23 |
| 10.1. Normative References . . . . .           | 23 |
| 10.2. Informative References . . . . .         | 23 |
| Appendix A. Profile example . . . . .          | 23 |
| Authors' Addresses . . . . .                   | 24 |

## 1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [I-D.ietf-core-link-format] and can be used by CoAP [I-D.ietf-core-coap] or HTTP servers. The CoRE Link Format defines an attribute that can be used to describe the REST interface of a resource, and may include a link to a description document. This memo describes how other specifications can combine resources with a well-known interface to create new CoRE RESTful profiles. A CoRE profile is based on the concept of Function Set, which is a group of REST resources providing a service in a distributed system. In addition, the notion of Binding is introduced in order to create a synchronization link between two resources. This document also defines well-known interface descriptions for Batch, Sensor, Parameter and Actuator types to compose new Function Sets or for standalone use in a constrained web server. A short reference is provided for each type that can be efficiently included in the interface description (if=) attribute of the CoRE Link Format.

## 2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [I-D.ietf-core-link-format]. This specification makes use of the following additional terminology:

**Function Set:** A group of well-known REST resources that provides a particular service.

**Profile:** A group of well-known Function Sets defined by a specification.

Device: An IP smart object running a web server that hosts a group of Function Set instances from a profile.

Service Discovery: The process making it possible for a web client to automatically detect devices and Function Sets offered by these devices on a CoRE network.

Resource Discovery: The process allowing a web client to identify resources being hosted on a web server.

Gradual Reveal: A REST design where resources are discovered progressively using Web Linking.

Binding: A unidirectional logical link between a source resource and a destination resource.

### 3. Function Set

This section defines how a specification can organize REST resources to create a new profile. A profile is structured into groups of resource types called Function Sets. A Function Set is similar to a function block in the sense that it consists of input, output and parameter resources and contains internal logic. A Function Set MAY have a subset of mandatory inputs, outputs and parameters to provide minimum interoperability. It MAY also be extended with manufacturer/user-specific resources. Other specifications defines the list of function sets available within a given profile. A device is composed of one or more Function Set instances. A profile specification MAY specify device profiles with mandatory function sets.

#### 3.1. Defining a Function Set

In a Function Set, types of resources are defined. Each type includes a human readable name, a path template, a Resource Type for discovery, the Interface Definition and the data type and allowed values. A Function Set definition may also include a field indicating if a sub-resource is mandatory or optional.

##### 3.1.1. Path template

A Function Set is a container resource under which its sub-resources are organized. The profile defines the path to each resource of a Function Set in a path template. The template can contain either relative paths or absolute paths depending on the profile needs. An absolute Function Set SHOULD be located at its recommended root path on a web server, however it MAY be located under an alternative path if necessary (for example multi-purpose devices, gateways etc.). A relative Function Set can be instantiated as many times as needed on

a web server with an arbitrary root path. However some Function Sets (e.g. device description) only make sense as singletons.

The path template includes a possible index {#} parameter, and possible fixed path segments. The index {#} allows for multiple instances of this type of resource, and can be any string. The root path and the indexes are the only variable elements in a path template. All other path segments MUST be fixed.

### 3.1.2. Resource Type

Each root resource of a Function Set is assigned a Resource Type parameter, therefore making it possible to discover it. Each sub-resource of a Function Set is also assigned a Resource Type parameter. This Resource Type is used for resource discovery and is usually necessary to discover optional resources supported on a specific device. The Resource Type of a Function Set may also be used for service discovery and MAY be exported to DNS-SD [I-D.cheshire-dnsext-dns-sd] for example.

The Resource Type parameter defines the value that MUST be included in the rt= field of the CoRE Link Format when describing a link to this resource. The value SHOULD be in the form "namespace.type" for root resources and "namespace.type.subtype" for sub-resources. This naming convention facilitates resource type filtering with the /.well-known/core resource. However a profile MAY allow mixing in foreign namespace references within a Function Set to import external references from other object models (e.g. SenML and UCUM).

### 3.1.3. Interface Description

The Interface Description parameter defines the REST interface for that type of resource. Several base interfaces are defined in Section 5 of this document. For a given profile, the Interface Description may be inferred from the Resource Type. In that case the Interface Description MAY be elided from link descriptions of resource types defined in the profile, but SHOULD be included for custom extensions to the profile.

The root resource of a Function Set should provide a list of links to its sub-resources in order to offer gradual reveal of resources. The CoRE Link List interface defined in Section 5.1 offers this functionality so a root resource SHOULD support this interface or a derived interface like CoRE Batch (See Section 5.2).

### 3.1.4. Data type

The Data Type field defines the type of value (and possible range) that is returned in response to a GET for that resource or accepted with a PUT. The interfaces defined in Section 5 make use of plain text and SenML Media types for the actual format of this data. A profile may restrict the list of supported content types for the CoRE interfaces or define new interfaces with new content types.

### 3.2. Discovery

A device conforming to a profile SHOULD make its resources discoverable by providing links to the resources on the path `/.well-known/core` as defined in [I-D.ietf-core-link-format]. All resources hosted on a device SHOULD be discoverable either with a direct link in `/.well-known/core` or by following successive links starting from `/.well-known/core`.

The root path of a Function Set instance SHOULD be directly referenced in `/.well-known/core` in order to offer discovery at the first discovery stage. A device with more than 10 individual resources SHOULD only expose Function Set instances in `/.well-known/core` to limit the size of this resource.

In addition, a device MAY register its resources to a Resource Directory using the registration interface defined in [I-D.shelby-core-resource-directory] if such a directory is available.

### 3.3. Versioning

A profile should track Function Set changes to avoid incompatibility issues. Evolutions in a Function Set SHOULD be backward compatible.

## 4. Bindings

In a M2M RESTful environment, endpoints exchange the content of their resources to operate the distributed system. Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human intervention and a so-called commissioning tool. In this document the abstract relationship between two resources is called a Binding. The configuration phase necessitates the exchange of binding information so a format recognized by all CoRE endpoints is essential. This document defines a format based on the CoRE Link-Format to represent binding information along with the rules to define a binding method which is a specialized relationship between two resources. The purpose of a binding is to synchronize the content between a source resource and a destination resource. The



destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method. The following table gives a summary of the binding methods described in more detail in Section 4.2.

| Name    | Identifier | Location    | Method        |
|---------|------------|-------------|---------------|
| Polling | poll       | Destination | GET           |
| Observe | obs        | Destination | GET + Observe |
| Push    | push       | Source      | PUT           |

#### 4.1. Format

Since Binding lies in the creation of a link between two resources, Web Linking and the CoRE Link-Format are a natural way to represent binding information. This involves the creation of a new relation type, purposely named "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource. The Web link attributes allow a fine-grained control of the type of synchronization exchange along with the conditions that trigger an update. This specification defines the attributes below:

| Attribute          | Parameter | Value            |
|--------------------|-----------|------------------|
| Binding method     | bind      | xsd:string       |
| Minimum Period (s) | pmin      | xsd:integer (>0) |
| Maximum Period (s) | pmax      | xsd:integer (>0) |
| Change Step        | st        | xsd:decimal (>0) |

**Bind Method:** This is the identifier of a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

**Minimum Period:** When present, the minimum period indicates the minimum time to wait (in seconds) before sending a new synchronization message (even if it has changed). In the absence of this parameter, the minimum period is up to the notifier.

Maximum Period: When present, the maximum period indicates the maximum time in seconds between two consecutive synchronization messages (regardless if it has changed). In the absence of this parameter, the maximum period is up to the notifier. The maximum period **MUST** be greater than the minimum period parameter (if present).

Change Step: When present, the change step indicates how much the value of a resource **SHOULD** change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification **SHOULD** be between pmin and pmax.

#### 4.2. Binding methods

A binding method defines the rules to generate the web-transfer exchanges that will effectively send content from the source resource to the destination resource. The description of a binding method must define the following aspects:

Identifier: This is value of the "bind" attribute used to identify the method.

Location: This information indicates whether the binding entry is stored on the source or on the destination endpoint.

REST Method: This is the REST method used in the Request/Response exchanges.

Conditions: A binding method definition must state how the condition attributes of the abstract binding definition are actually used in this specialized binding.

This specification supports 3 binding methods described below.

Polling: The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method **MUST** be stored on the destination endpoint. The destination endpoint **MUST** ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process **MAY** filter out content from the GET requests using value-based conditions (e.g Change Step).

Observe: The Observe method relies on the Publish/Subscribe pattern thus an observation relationship is created between the

destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [I-D.ietf-core-observe] hence this method is only permitted when the resources are made available over CoAP. The binding entry for this method MUST be stored on the destination endpoint. The binding conditions are mapped as query string parameters (see Section 5.9).

Push: When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource upon change of the source resource. The source endpoint MUST only send a notification request if the binding conditions are met. The binding entry for this method MUST be stored on the source endpoint.

#### 4.3. Binding table

The binding table is a special resource that gives access to the bindings on a endpoint. A binding table resource MUST support the Binding interface defined in Section 5.8. A profile SHOULD allow only one resource table per endpoint.

### 5. Interface Descriptions

This section defines REST interfaces for Link List, Batch, Sensor, Parameter, Actuator and Binding table resources. Variants such as Linked Batch or Read-Only Parameter are also presented. Each type is described along with its Interface Description attribute value and valid methods. These are defined for each interface in the table below. These interfaces can support plain text and/or SenML Media types.

The if= column defines the Interface Description (if=) attribute value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this specification. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

The Methods column defines the methods supported by that interface, which are described in more detail below.

| Interface           | if=      | Methods                                   |
|---------------------|----------|---|
| Link List           | core.ll  | GET                                       |
| Batch               | core.b   | GET, PUT, POST (where applicable)         |
| Linked Batch        | core.lb  | GET, PUT, POST, DELETE (where applicable) |
| Sensor              | core.s   | GET                                       |
| Parameter           | core.p   | GET, PUT                                  |
| Read-only Parameter | core.rp  | GET                                       |
| Actuator            | core.a   | GET, PUT, POST                            |
| Binding             | core.bnd | GET, POST, DELETE                         |

The following is an example of links in the CoRE Link Format using these interface descriptions. The resource hierarchy is based on a simple profile defined in Appendix A. These links are used in the subsequent examples below.

```

Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s>;rt="simple.sen";if="core.b",
</s>;rt="simple.sen.lt";if="core.s",
</s>;rt="simple.sen.tmp";if="core.s";obs,
</s>;rt="simple.sen.hum";if="core.s",
</a>;rt="simple.act";if="core.b",
</a>;rt="simple.act.led";if="core.a",
</a>;rt="simple.act.led";if="core.a",
</d>;rt="simple.dev";if="core.ll",
</l>;if="core.lb",

```

### 5.1. Link List

The Link List interface is used to retrieve (GET) a list of resources on a web server. The GET request SHOULD contain an Accept option with the application/link-format content type, however if the resource does not support any other form of GET methods the Accept option MAY be elided. The Accept option SHOULD only include the application/link-format content type. The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on a web server.

Link List is the base interface to provide gradual reveal of resources on a CoRE web server, hence the root resource of a Function Set SHOULD implement this interface or an extension of this interface.

The following example interacts with a Link List /d containing Parameter sub-resources /d/name, /d/model.

```
Req: GET /d (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"
```

## 5.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface type supports the same methods as its sub-resources, and can be used to read (GET), set (PUT) or toggle (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous, a method used on the Batch only applies to sub-resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

In addition, The Batch interface is an extension of the Link List interface and in consequence MUST support the same methods.

The following example interacts with a Batch /s with Sensor sub-resources /s/light, /s/temp and /s/humidity.

```
Req: GET /s
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "light", "v": 123, "u": "lx" },
  { "n": "temp", "v": 27.2, "u": "degC" },
  { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

## 5.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. 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 [I-D.ietf-core-link-format]. A request with a POST method and a content type of application/link-format simply appends new resources to the collection. The links in the payload MUST reference a resource on the web server with an absolute path. A DELETE request empties the current collection of links. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.

```
Req: POST /l (Content-type: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed
```

```
Req: GET /l
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "degC" },
]
```

```
Req: POST /l (Content-type: application/link-format)
</s/humidity>
Res: 2.04 Changed
```

```
Req: GET /l (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>
```

```
Req: GET /l
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "degC" },
  { "n": "/s/humidity", "v": 80, "u": "%RH" }],
}
```

```
Req: DELETE /l
Res: 2.04 Changed
```

#### 5.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80
```

```
Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
{"e":[
  { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

#### 5.5. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or set (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and setting a parameter.

```
Req: GET /d/name
Res: 2.05 Content (text/plain)
node5
```

```
Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed
```

### 5.6. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not set. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

### 5.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or a new actuator value set (PUT). In addition, this interface defines the use of POST (with no body) to toggle an actuator between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to set.

The following example shows requests for reading, setting and toggling an actuator (turning on a led).



```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

```
Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed
```

```
Req: POST /a/1/led (text/plain)
Res: 2.04 Changed
```

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

## 5.8. Binding

The Binding interface is used to manipulate a binding table. A request with a POST method and a content type of application/link-format simply appends new bindings to the table. All links in the payload MUST have a relation type "boundTo". A GET request simply returns the current state of a binding table whereas a DELETE request empties the table.

The following example shows requests for adding, retrieving and deleting bindings in a binding table.

```
Req: POST /bnd (Content-type: application/link-format)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
Res: 2.04 Changed
```

```
Req: GET /bnd
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
```

```
Req: DELETE /bnd
Res: 2.04 Changed
```

## 5.9. Resource Observation Attributes

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [I-D.ietf-core-observe] MAY be used to observe any changes in a resource, and receive asynchronous notifications as a result. In addition, a set of query string parameters are defined here to allow a client to control how often a client is interested in receiving notifications and how much a resource value should change for the new representation to be interesting. These query parameters are described in the following table. A resource using an interface description defined in this specification and marked as Observable in its link description SHOULD support these observation parameters. The Change Step parameter can only be supported on resources with an atomic numeric value.

These query parameters MUST be treated as resources that are read using GET and set using PUT, and MUST NOT be included in the Observe request. Multiple parameters MAY be set at the same time by including the values in the query string of a PUT. Before being set, these parameters have no default value.

| Resource           | Parameter          | Data Format      |
|--------------------|--------------------|------------------|
| Minimum Period (s) | / {resource} ?pmin | xsd:integer (>0) |
| Maximum Period (s) | / {resource} ?pmax | xsd:integer (>0) |
| Change Step        | / {resource} ?st   | xsd:decimal (>0) |
| Less Than          | / {resource} ?lt   | xsd:decimal      |
| Greater Than       | / {resource} ?gt   | xsd:decimal      |

**Minimum Period:** When set, the minimum period indicates the minimum time in seconds the server SHOULD wait between sending notifications. In the absence of this parameter, the minimum period is up to the server.

**Maximum Period:** When set, the maximum period indicated the maximum time in seconds the server SHOULD wait between sending notifications (regardless if it has changed). In the absence of this parameter, the maximum period is up to the server. The maximum period MUST be greater than the minimum period parameter (if present).

**Change Step:** When set, the change step indicates how much the value of a resource SHOULD change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification SHOULD be between pmin and pmax.

**Less Than:** When set, the value of the resource MUST be less than this parameter in order to send a new notification. This parameter has lower priority than the period parameters.

**Greater Than:** When set, the value of the resource MUST be greater than this parameter in order to send a new notification. This parameter has lower priority than the period parameters.

The following example shows an Observation request using these query parameters. Here the value of Observe indicates the number of seconds since the observation was made to show the time.

Req PUT /s/temp?pmin=10&pmax=60&st=1

Res: 2.04 Changed

Req: GET Observe /s/temp

Res: 2.05 Content Observe:0 (text/plain)  
23.2

Res: 2.05 Content Observe:60 (text/plain)  
23.0

Res: 2.05 Content Observe:80 (text/plain)  
22.0

Res: 2.05 Content Observe:140 (text/plain)  
21.8

The following example shows a request to check the current value of the pmin attribute of the Observable resource from the last example.

Req: GET /s/temp?pmin

Res: 2.05 Content (text/plain)  
10

### 5.10. Future Interfaces

It is expected that further interface descriptions will be defined in this and other specifications. Potential interfaces to be considered for this specifications include:

Collection: This resource would be a container that allows sub-resources to be added or removed.

### 5.11. WADL Description

This section defines the formal Web Application Description Language (WADL) definition of these CoRE interface descriptions.

```
<?xml version="1.0" standalone="yes"?>

<application xmlns="http://research.sun.com/wadl/2006/10"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:senml="urn:ietf:params:xml:ns:senml">

  <grammars>
    <include href="http://tools.ietf.org/html/draft-jennings-senml"/>
  </grammars>

  <doc title="CoRE Interfaces"/>

  <resource_type id="s">
    <doc title="Sensor resource type"/>
    <method href="#read"/>
    <method href="#observe"/>
  </resource_type>

  <resource_type id="p">
    <doc title="Parameter resource type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#update"/>
  </resource_type>

  <resource_type id="rp">
    <doc title="Read-only Parameter resource type"/>
    <method href="#read"/>
    <method href="#observe"/>
  </resource_type>

  <resource_type id="a">
    <doc title="Actuator resource type"/>
```

```
<method href="#read"/>
<method href="#observe"/>
<method href="#update"/>
<method href="#toggle"/>
</resource_type>

<resource_type id="ll">
  <doc title="Link List type"/></doc>
  <method href="#listLinks"/>
</resource_type>

<resource_type id="b">
  <doc title="Batch of sub-resources type">The methods read,
    observe, update and toggle are applied to each sub-
    resource of the requested resource that supports it. Mixed
    sub-resource types can be supported.</doc>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#update"/>
  <method href="#toggle"/>
  <method href="#listLinks"/>
</resource_type>

<resource_type id="lb">
  <doc title="Linked Batch resource type">. The methods read,
    observableRead, update and toggle are applied to each linked
    resource of the requested resource that supports it. Mixed
    linked resource types can be supported.</doc>
  <method href="#read"/>
  <method href="#observe"/>
  <method href="#update"/>
  <method href="#listLinks"/>
  <method href="#appendLinks"/>
  <method href="#clearLinks"/>
</resource_type>

<resource_type id="bnd">
  <doc title="Binding table resource type">A modifiable list of
    links. Each link MUST have the relation type "boundTo".</doc>
  <method href="#listLinks"/>
  <method href="#appendLinks"/>
  <method href="#clearLinks"/>
</resource_type>

<method id="read" name="GET">
  <doc>Retrieve the value of a sensor, an actuator or a parameter.
    Both HTTP and CoAP support this method.</doc>
  <request>
```

```
</request>
<response status="200">
  <representation mediaType="text/plain"/>
  <representation mediaType="application/senml+exi"/>
  <representation mediaType="application/senml+xml"/>
  <representation mediaType="application/senml+json"/>
</response>
<response status="2.05">
  <representation mediaType="text/plain"/>
  <representation mediaType="application/senml+exi"/>
  <representation mediaType="application/senml+xml"/>
  <representation mediaType="application/senml+json"/>
</response>
</method>

<method id="observe" name="GET">
  <doc>Observe the value of a sensor, an actuator or a parameter.
  Only CoAP supports this method since it requires the CoRE
  Observe mechanism.</doc>
  <request>
    <param name="pmin" style="query" type="xsd:integer"/>
    <param name="pmax" style="query" type="xsd:integer"/>
    <param name="st" style="query" type="xsd:decimal"/>
  </request>
  <response status="2.05">
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </response>
</method>

<method id="update" name="PUT">
  <doc>Control the actuator or update a parameter with a new value
  or command. Both HTTP and CoAP support this method.</doc>
  <request>
    <representation mediaType="text/plain"/>
    <representation mediaType="application/senml+exi"/>
    <representation mediaType="application/senml+xml"/>
    <representation mediaType="application/senml+json"/>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="toggle" name="POST">
  <doc>Toggle the values of actuator resources. Both HTTP and CoAP
  support this method.</doc>
```

```
<request>
  <doc>The toggle function is only applicable if the request
    is empty.</doc>
</request>
<response status="200"/>
<response status="2.04"/>
</method>

<method id="listLinks" name="GET">
  <doc>Retrieve the list of Web links associated to a resource.
    Both HTTP and CoAP support this method.</doc>
  <request>
    <doc>This request MUST contain an Accept option with
      application/link-format when the resource supports
      other GET methods.</doc>
  </request>
  <response status="200">
    <representation mediaType="application/link-format"/>
  </response>
  <response status="2.05">
    <representation mediaType="application/link-format"/>
  </response>
</method>

<method id="appendLinks" name="POST">
  <doc>Append new Web links to a resource which is a collection
    of links. Both HTTP and CoAP support this method.</doc>
  <request>
    <representation mediaType="application/link-format"/>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

<method id="clearLinks" name="DELETE">
  <doc>Clear all Web Links in a resource which is a collection
    of links. Both HTTP and CoAP support this method.</doc>
  <request>
  </request>
  <response status="200"/>
  <response status="2.04"/>
</method>

</application>
```

## 6. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service.

## 7. IANA Considerations

The interface description types defined require registration.

The new link relation type "boundto" requires registration.

## 8. Acknowledgments

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document.

## 9. Changelog

Changes from -04 to -05

- o Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- o Draft refresh

Changes from -02 to -03

- o Added Bindings
- o Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- o Defined a Function Set and its guidelines.



- o Added the Link List interface.
- o Added the Linked Batch interface.
- o Improved the WADL interface definition.
- o Added a simple profile example.

## 10. References

### 10.1. Normative References

- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-14 (work in progress), June 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

### 10.2. Informative References

- [I-D.cheshire-dnsext-dns-sd]  
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-11 (work in progress), December 2011.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-14 (work in progress), March 2013.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.
- [I-D.shelby-core-resource-directory]  
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-05 (work in progress), February 2013.

## Appendix A. Profile example

The following is a short definition of simple profile. This simplistic profile is for use in the examples of this document.

| Function Set       | Root Path | RT         | IF      |
|--------------------|-----------|------------|---------|
| Device Description | /d        | simple.dev | core.ll |
| Sensors            | /s        | simple.sen | core.b  |
| Actuators          | /a        | simple.act | core.b  |

## List of Function Sets

| Type  | Path     | RT             | IF      | Data Type  |
|-------|----------|----------------|---------|------------|
| Name  | /d/name  | simple.dev.n   | core.p  | xsd:string |
| Model | /d/model | simple.dev.mdl | core.rp | xsd:string |

## Device Description Function Set

| Type        | Path        | RT             | IF     | Data Type             |
|-------------|-------------|----------------|--------|-----------------------|
| Light       | /s/light    | simple.sen.lt  | core.s | xsd:decimal<br>(lux)  |
| Humidity    | /s/humidity | simple.sen.hum | core.s | xsd:decimal<br>(%RH)  |
| Temperature | /s/temp     | simple.sen.tmp | core.s | xsd:decimal<br>(degC) |

## Sensors Function Set

| Type | Path       | RT             | IF     | Data Type   |
|------|------------|----------------|--------|-------------|
| LED  | /a/{#}/led | simple.act.led | core.a | xsd:boolean |

## Actuators Function Set

## Authors' Addresses

Zach Shelby  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: zach@sensinode.com

Matthieu Vial  
Schneider-Electric  
Grenoble  
FRANCE

Phone: +33 (0)47657 6522  
Email: matthieu.vial@schneider-electric.com

CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: August 29, 2013

Z. Shelby  
Sensinode  
S. Krco  
Ericsson  
C. Bormann  
Universitaet Bremen TZI  
February 25, 2013

CoRE Resource Directory  
draft-shelby-core-resource-directory-05

Abstract

In many M2M applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources. This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to register, maintain, lookup and remove resources descriptions. Furthermore, new link attributes useful in conjunction with an RD are defined.

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 August 29, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

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

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .                             | 3  |
| 2. Terminology . . . . .                              | 3  |
| 3. Architecture and Use Cases . . . . .               | 4  |
| 3.1. Use Case: Cellular M2M . . . . .                 | 5  |
| 3.2. Use Case: Home and Building Automation . . . . . | 6  |
| 4. Simple Directory Discovery . . . . .               | 6  |
| 4.1. Finding a Directory Server . . . . .             | 7  |
| 5. Resource Directory Function Set . . . . .          | 8  |
| 5.1. Discovery . . . . .                              | 8  |
| 5.2. Registration . . . . .                           | 10 |
| 5.3. Update . . . . .                                 | 12 |
| 5.4. Validation . . . . .                             | 13 |
| 5.5. Removal . . . . .                                | 15 |
| 6. Group Function Set . . . . .                       | 16 |
| 6.1. Register a Group . . . . .                       | 16 |
| 6.2. Group Removal . . . . .                          | 17 |
| 7. RD Lookup Function Set . . . . .                   | 18 |
| 8. New Link-Format Attributes . . . . .               | 23 |
| 8.1. Resource Instance 'ins' attribute . . . . .      | 23 |
| 8.2. Export 'exp' attribute . . . . .                 | 23 |
| 9. Security Considerations . . . . .                  | 24 |
| 10. IANA Considerations . . . . .                     | 24 |
| 11. Acknowledgments . . . . .                         | 24 |
| 12. Changelog . . . . .                               | 24 |
| 13. References . . . . .                              | 26 |
| 13.1. Normative References . . . . .                  | 26 |
| 13.2. Informative References . . . . .                | 26 |
| Authors' Addresses . . . . .                          | 26 |

## 1. Introduction

The Constrained RESTful Environments (CoRE) work aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. This specification however only describes how to discover resources from the web server that hosts them by requesting /.well-known/core. In many M2M scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which hosts descriptions of resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports in order for web servers to discover the RD and to registrar, maintain, lookup and remove resource descriptions. Furthermore, new link attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [I-D.ietf-core-coap], they may be applied in an equivalent manner to HTTP [RFC2616].

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [I-D.ietf-core-coap]. The URI Template format is used to describe the REST interfaces defined in this specification [RFC6570]. This specification makes use of the following additional terminology:

#### Resource Directory

An web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

#### Domain

In the context of a Resource Directory, a domain is a logical grouping of endpoints. All endpoint within a domain MUST be unique. This specification assumes that the list of Domains supported by an RD is pre-configured by that RD.

#### Group

In the context of a Resource Directory, a group is a logical grouping of endpoints for the purpose of group communications. All groups within a domain MUST be unique.

#### Endpoint

An endpoint (EP) is a term used to describe a web server or client in [I-D.ietf-core-coap]. In the context of this specification an endpoint is used to describe a web server that registers resources to the Resource Directory. An endpoint is identified by its endpoint name, which is included during registration, and MUST be unique within the associated domain of the registration.

### 3. Architecture and Use Cases

The resource directory architecture is shown in Figure 1. A Resource Directory (RD) is used as a repository for Web Links [RFC5988] about resources hosted on other web servers, which are called endpoints (EP). An endpoint is a web server associated with a port, thus a physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain sets of Web Links (called resource directory entries), for the RD to validate entries, and for clients to lookup resources from the RD. Endpoints themselves can also act as clients. An RD can be logically segmented by the use of Domains. The domain an endpoint is associated with can be defined by the RD or configured by an outside entity.

Endpoints are assumed to proactively register and maintain resource directory entries on the RD, which are soft state and need to be periodically refreshed. An endpoint is provided with interfaces to register, update and remove a resource directory entry. Furthermore, a mechanism to discover a RD using the CoRE Link Format is defined. It is also possible for an RD to proactively discover Web Links from endpoints and add them as resource directory entries, or to validate existing resource directory entries. A lookup interface for discovering any of the Web Links held in the RD is provided using the

CoRE Link Format.

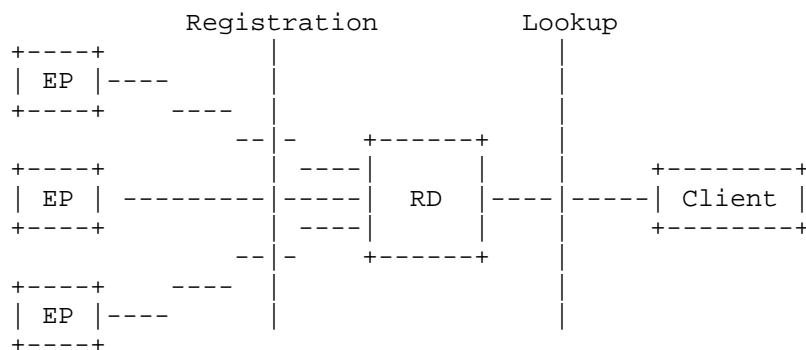


Figure 1: The resource directory architecture.

### 3.1. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users, i.e. machines. The machines are connected directly to a mobile network using appropriate embedded air interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines - endpoints) capable of providing required information at a given time or acting on instructions from the end users.

In a typical scenario, during a boot-up procedure (and periodically afterwards), the machines (endpoints) register with a Resource Directory (for example EPs installed on vehicles enabling tracking of their position for the fleet management purposes and monitoring environment parameters) hosted by the mobile operator or somewhere else in the network, submitting a description of own capabilities. Due to the usual network configuration of mobile networks, the EPs attached to the mobile network do not have routable addresses. Therefore, a remote server is usually used to provide proxy access to the EPs. The address of each (proxy) endpoint on this server is included in the resource description stored in the RD. The users, for example mobile applications for environment monitoring, contact the RD, look-up the endpoints capable of providing information about



the environment using appropriate set of tags, obtain information on how to contact them (URLs of the proxy server) and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide a set of credentials along with the appropriate tags to the RD to look-up for EPs deployed on the vehicles the application is responsible for.

### 3.2. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of M2M web services. The use of CoRE in home automation across multiple subnets is described in [I-D.brandt-coap-subnet-discovery] and in commercial building automation in [I-D.vanderstok-core-bc]. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices.

The exporting of resource information to other discovery systems is also important in these automation applications. In home automation there is a need to interact with other consumer electronics, which may already support DNS-SD, and in building automation larger resource directories or DNS-SD covering multiple buildings.

## 4. Simple Directory Discovery

Not all endpoints hosting resources are expected to know how to implement the Resource Directory Function Set and thus explicitly register with a Resource Directory (or other such directory server). Instead, simple endpoints can implement the generic Simple Directory Discovery approach described in this section. An RD implementing this specification MUST implement Simple Directory Discovery. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the endpoint makes the hosted resources that it wants discovered available as links on its /.well-known/core interface as specified in [RFC6690].

The endpoint then finds one or more IP addresses of the directory server it wants to know about its resources as described in Section 4.1.

An endpoint that wants to make itself discoverable occasionally sends

a POST request to the /.well-known/core URI of any candidate directory server that it finds. The body of the POST request is either

- o empty, in which case the directory server is encouraged by this POST request to perform GET requests at the requesting server's default discovery URI.

or

- o a link-format document, which indicates the specific services that the requesting server wants to make known to the directory server.

The directory server integrates the information it received this way into its resource directory. It MAY make the information available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

The following example shows an endpoint using simple resource discovery, by simply sending a POST with its links in the body to a directory.

```
EP                                     RD
|                                     |
| -- POST /.well-known/core "</sen/temp>..." ----> |
|                                     |
| <----- 2.01 Created -----> |
|                                     |
```

#### 4.1. Finding a Directory Server

Endpoints that want to contact a directory server can obtain candidate IP addresses for such servers in a number of ways.

In a 6LoWPAN, good candidates can be taken from:

- o specific static configuration (e.g., anycast addresses), if any,
- o the ABRO option of 6LoWPAN-ND [RFC6775],
- o other ND options that happen to point to servers (such as RDNSS),

- o DHCPv6 options that might be defined later.

In networks with more inexpensive use of multicast, the candidate IP address may be a well-known multicast address, i.e. directory servers are found by simply sending POST requests to that well-known multicast address (details TBD).

As some of these sources are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. E.g., an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

## 5. Resource Directory Function Set

This section defines the REST interfaces between an RD and endpoint servers, which is called the Resource Directory Function Set. Although the examples throughout this section assume use of CoAP [I-D.ietf-core-coap], these REST interfaces can also be realized using HTTP [RFC2616]. An RD implementing this specification MUST support the discovery, registration, update, and removal interfaces defined in this section and MAY support the validation interface. For the purpose of validation, an endpoint implementing this specification SHOULD support ETag validation on /.well-known/core (which is very straightforward for static /.well-known/core link documents).

Resource directory entries are designed to be easily exported to other discovery mechanisms such as DNS-SD. For that reason, parameters that would meaningfully be mapped to DNS are limited to a maximum length of 63 bytes.

### 5.1. Discovery

Before an endpoint can make use of an RD, it must first know the RD's IP address, port and the path of its RD Function Set. There can be several mechanisms for discovering the RD including assuming a default location (e.g. on an Edge Router in a LoWPAN), by assigning an anycast address to the RD, using DHCP, or by discovering the RD using the CoRE Link Format (also see Section 4.1). This section defines discovery of the RD using the well-known interface of the CoRE Link Format [RFC6690] as the required mechanism. It is however expected that RDs will also be discoverable via other methods depending on the deployment.

Discovery is performed by sending either a multicast or unicast GET request to `/.well-known/core` and including a Resource Type (rt) parameter [RFC6690] with the value `"core.rd"` in the query string. Likewise, a Resource Type parameter value of `"core.rd-lookup"` is used to discover the RD Lookup Function Set. Upon success, the response will contain a payload with a link format entry for each RD discovered, with the URL indicating the root resource of the RD. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network.

An RD implementation of this specification MUST support query filtering for the rt parameter as defined in [RFC6690].

The discovery request interface is specified as follows:

Interaction: EP -> RD

Method: GET

URI Template: `/.well-known/core{?rt}`

URI Template Variables:

rt := Resource Type (optional). MAY contain the value `"core.rd"`, `"core.rd-lookup"` or `"core.rd*"`

Content-Type: `application/link-format` (if any)

The following response codes are defined for this interface:

Success: 2.05 "Content" with an `application/link-format` payload containing a matching entry for the RD resource.

Failure: 4.04 "Not Found" is returned in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request"

The following example shows an endpoint discovering an RD using this interface, thus learning that the base RD resource is at `/rd`. Note that it is up to the RD to choose its base RD resource, although it is recommended to use default locations where possible.

```

      EP                                     RD
      |                                     |
      |----- GET /.well-known/core?rt=core.rd* ----->
      |
      |<----- 2.05 Content "</rd>; rt="core.rd" -----
      |

```

Req: GET coap://[ff02::1]/.well-known/core?rt=core.rd\*

Res: 2.05 Content  
 </rd>;rt="core.rd",  
 </rd-lookup>;rt="core.rd-lookup",  
 </rd-group>;rt="core.rd-group"

## 5.2. Registration

After discovering the location of an RD Function Set, an endpoint MAY register its resources using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format along with query string parameters indicating the name of the endpoint, its domain and the lifetime of the registration. All parameters except the endpoint name are optional. It is expected that other specifications MAY define further parameters (it is to be determined if a registry of parameters is needed for this purpose). The RD then creates a new resource or updates an existing resource in the RD and returns its location. An endpoint MUST use that location when refreshing registrations using this interface. Endpoint resources in the RD are kept active for the period indicated by the lifetime parameter. The endpoint is responsible for refreshing the entry within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameter does not create multiple RD entries.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: `/{"rd"}{"?ep,d,et,lt,con"}`

URI Template Variables:

`rd` := RD Function Set path (mandatory). This is the path of the RD Function Set. An RD SHOULD use the value "rd" for this variable whenever possible.

`ep` := Endpoint (mandatory). The endpoint identifier or name of the registering node, unique within that domain. The maximum length of this parameter is 63 bytes.

`d` := Domain (optional). The domain to which this endpoint belongs. The maximum length of this parameter is 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain.

`et` := Endpoint Type (optional). The semantic type of the endpoint. The maximum length of this parameter is 63 bytes. Optional.

`lt` := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

`con` := Context (optional). This parameter sets the scheme, address and port at which this server is available in the form `scheme://host:port`. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port of the register request are assumed.

Content-Type: `application/link-format`

The following response codes are defined for this interface:

Success: 2.01 "Created". The Location header MUST be included with the new resource entry for the endpoint. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration (update, delete).

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an endpoint with the name "node1" registering two resources to an RD using this interface. The resulting location `/rd/4521` is just an example of an RD generated

location.

```

      EP                                     RD
      |                                     |
      | --- POST /rd?ep=node1 "</sensors..." ----->
      |
      | <-- 2.01 Created Location: /rd/4521 -----
      |

```

Req: POST coap://rd.example.com/rd?ep=node1

Payload:

```
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
</sensors/light>;ct=41;rt="light-lux";if="sensor"
```

Res: 2.01 Created

Location: /rd/4521

### 5.3. Update

The update interface is used by an endpoint to refresh or update its registration with an RD. To use the interface, the endpoint sends a PUT request to the resource returned in the Location option in the response to the first registration. An update MAY contain registration parameters if there have been changes since the last registration or update. Parameters that have not changed SHOULD NOT be included in an update. Upon receiving an update request, the RD resets the timeout for that endpoint and stores the values of the parameters included in the update (if any).

The update request interface is specified as follows:

Interaction: EP -> RD

Method: PUT

URI Template: /{+location}{?et,lt,con}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful registration.

et := Endpoint Type (optional). The semantic type of the endpoint. The maximum length of this parameter is 63 bytes. Optional.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, a default value of 86400 (24 hours) SHOULD be assumed.

con := Context (optional). This parameter sets the scheme, address and port at which this server is available in the form scheme://host:port. Optional. In the absence of this parameter the scheme of the protocol, source IP address and source port used to register are assumed.

Content-Type: None

The following response codes are defined for this interface:

Success: 2.04 "Changed" in the update was successfully processed.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows an endpoint updating a new set of resources to an RD using this interface.



Req: PUT /rd/4521

Res: 2.04 Changed

#### 5.4. Validation

In some cases, an RD may want to validate that it has the latest version of an endpoint's resources. This can be performed with a GET on the well-known interface of the CoRE Link Format including the



latest ETag stored for that endpoint. For the purpose of validation, an endpoint implementing this specification SHOULD support ETag validation on /.well-known/core.

The validation request interface is specified as follows:

Interaction: RD -> EP

Method: GET

Path: /.well-known/core

Parameters: None

ETag: The ETag option MUST be included

The following responses codes are defined for this interface:

Success: 2.03 "Valid" in case the ETag matches

Success: 2.05 "Content" in case the ETag does not match, the response MUST include the most recent resource representation (application/link-format) and its corresponding ETag.

Failure: 4.00 "Bad Request". Malformed request.

The following examples shows a successful validation.

| EP  | RD |
|---|----|
|   |    |
| <--- GET /.well-known/core ETag: 0x40 ----- |    |
|   |    |
| --- 2.03 Valid ----->                       |    |
|   |    |

Req: GET /.well-known/core  
ETag: 0x40

Res: 2.03 Valid

## 5.5. Removal

Although RD entries have soft state and will eventually timeout after their lifetime, an endpoint **SHOULD** explicitly remove its entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a **DELETE** on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: **DELETE**

URI Template: `/[+location]`

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful registration.

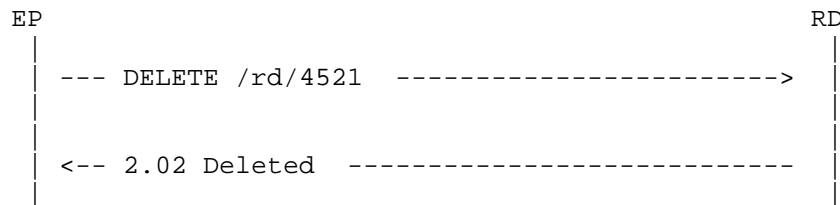
The following responses codes are defined for this interface:

Success: 2.02 "Deleted" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the endpoint from the RD.



Req: **DELETE** `/rd/4521`

Res: 2.02 Deleted

## 6. Group Function Set

This section defines a function set for the creation of groups of endpoints for the purpose of managing and looking up endpoints for group operations. The group function set is similar to the resource directory function set, in that a group may be created or removed. However unlike an endpoint entry, a group entry consists of a list of endpoints and does not have a lifetime associated with it. In order to make use of multicast requests with CoAP, a group MAY have a multicast address associated with it.

### 6.1. Register a Group

In order to create a group, a management entity used to configure groups, makes a request to the RD indicating the name of the group to create (or update), the optional domain the group belongs to, and the optional multicast address of the group. The registration message includes the list of endpoints that belong to that group. If an endpoint has already registered with the RD, the RD attempts to use the context of the endpoint from its RD endpoint entry. If the client registering the group knows the endpoint has already registered, then it MAY send a blank target URI for that endpoint link when registering the group.

The registration request interface is specified as follows:

Interaction: Manager -> RD

Method: POST

URI Template: `/{"rd-group"}{"?gp,d,con"}`

URI Template Variables:

`rd-group` := RD Group Function Set path (mandatory). This is the path of the RD Group Function Set. An RD SHOULD use the value "rd-group" for this variable whenever possible.

`gp` := Group Name (mandatory). The name of the group to be created or replaced, unique within that domain. The maximum length of this parameter is 63 bytes.

`d` := Domain (optional). The domain to which this group belongs. The maximum length of this parameter is 63 bytes. Optional. When this parameter is elided, the RD MAY associate the endpoint with a configured default domain.

**con** := Context (optional). This parameter is used to set the IP multicast address at which this server is available in the form scheme://multicast-address:port. Optional. In the absence of this parameter no multicast address is configured.

Content-Type: application/link-format

The following response codes are defined for this interface:

Success: 2.01 "Created". The Location header MUST be included with the new group entry. This Location MUST be a stable identifier generated by the RD as it is used for delete operations on this registration.

```
Failure: 4.00 "Bad Request". Malformed request.
```

```
Failure: 5.03 "Service Unavailable". Service could not perform the
operation.
```

The following example shows a group with the name "lights" registering two endpoints to an RD using this interface. The resulting location `/rd-group/12` is just an example of an RD generated group location.

| EP | RD  |
|----|---|
|    | - POST /rd-group?gp=lights "<>;ep=node1..." --> |
|    | <---- 2.01 Created Location: /rd-group/12 ----  |

```
Req: POST coap://rd.example.com/rd-group?gp=lights
```

Payload:

```
<> ; ep="node1",
```

```
<> ; ep= "node2 "
```

Res: 2.01 Created

Location: /rd-group/12

## 6.2. Group Removal

A group can be removed simply by sending a removal message to the location returned when registering the group. Removing a group **MUST NOT** remove the endpoints of the group from the RD.

The removal request interface is specified as follows:

Interaction: Manager -> RD

Method: DELETE

URI Template: /{+location}

URI Template Variables:

location := This is the Location path returned by the RD as a result of a successful group registration.

The following responses codes are defined for this interface:

Success: 2.02 "Deleted" upon successful deletion

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following examples shows successful removal of the group from the RD.

```

EP                                     RD
|                                     |
| --- DELETE /rd-group/412 -----> |
|                                     |
| <-- 2.02 Deleted -----         |
|                                     |

```

Req: DELETE /rd-group/12

Res: 2.02 Deleted

## 7. RD Lookup Function Set

In order for an RD to be used for discovering resources registered with it, a lookup interface can be provided using this function set. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. Atom or HTML Link) or using more advanced

interfaces (e.g. supporting context or semantic based lookup).

This function set allows lookups for domains, groups, endpoints and resources using attributes defined in the RD Function Set and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) in CoRE Link Format corresponding to the type of lookup. The target of these links SHOULD be the actual location of the domain, endpoint or resource, but MAY be an intermediate proxy e.g. in the case of an HTTP lookup interface for CoAP endpoints. Multiple query parameters MAY be included in a lookup, all included parameters MUST match for a resource to be returned. The character '\*' MAY be included at the end of a parameter value as a wildcard operator.

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: `/[+rd-lookup-base]/  
{lookup-type}{?d,ep,gp,et,rt,page,count,resource-param}`

Parameters:

`rd-lookup-base` := RD Lookup Function Set path (mandatory). This is the path of the RD Lookup Function Set. An RD SHOULD use the value "rd-lookup" for this variable whenever possible.

`lookup-type` := ("d", "ep", "res", "gp") (mandatory) This variable is used to select the kind of lookup to perform (domain, endpoint or resource).

`ep` := Endpoint (optional). Used for endpoint, group and resource lookups.

`d` := Domain (optional). Used for domain, group, endpoint and resource lookups.

`page` := Page (optional). Parameter can not be used without the count parameter. Results are returned from result set in pages that contains 'count' results starting from index (page \* count).

`count` := Count (optional). Number of results is limited to this parameter value. If the parameter is not present, then an RD implementation specific default value SHOULD be used.

rt := Resource type (optional). Used for group, endpoint and resource lookups.

rt := Endpoint type (optional). Used for group, endpoint and resource lookups.

resource-param := Link attribute parameters (optional). Any link attribute as defined in Section 4.1 of [RFC6690], used for resource lookups.

The following responses codes are defined for this interface:

Success: 2.05 "Content" with an application/link-format payload containing a matching entries for the lookup.

Failure: 4.04 "Not Found" in case no matching entry is found for a unicast request.

Failure: No error response to a multicast request.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows a client performing a resource lookup:

|        |  |    |
|--------|--|----|
| Client | <pre> ----- GET /rd-lookup/res?rt=temperature -----&gt;  &lt;-- 2.05 Content "&lt;coap://{ip:port}/temp" ---- </pre> | RD |
|--------|--|----|

Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content  
<coap://{ip:port}/temp>

The following example shows a client performing an endpoint lookup:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/ep?et=power-node ----->      |
|                                                         |
|<-- 2.05 Content "<coap://{ip:port}>;ep="node5" -----|
|                                                         |

```

Req: GET /rd-lookup/ep?et=power-node

Res: 2.05 Content  
 <coap://{ip:port}>;ep="node5",  
 <coap://{ip:port}>;ep="node7"

The following example shows a client performing a domain lookup:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/d ----->                      |
|                                                         |
|<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----|
|                                                         |

```

Req: GET /rd-lookup/d

Res: 2.05 Content  
 </rd>;d="domain1",  
 </rd>;d="domain2"

The following example shows a client performing a group lookup for all groups:

```

Client                                                     RD
|                                                         |
|----- GET /rd-lookup/gp ----->                    |
|                                                         |

```



```
<-- 2.05 Content </rd-group/12>;gp="lights1";d="domain1" --
```

```
Req: GET /rd-lookup/gp
```

```
Res: 2.05 Content
</rd-group/12>;gp="lights1";d="domain1"
```

The following example shows a client performing a lookup for all endpoints in a particular group:

| Client  | RD |
|---|----|
| ----- GET GET /rd-lookup/ep?gp=lights1----->            |    |
| <-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 ----- |    |

```
Req: GET /rd-lookup/ep?gp=lights1
```

```
Res: 2.05 Content
<coap://host:port>;ep="node1",
<coap://host:port>;ep="node2",
```

The following example shows a client performing a lookup for all groups an endpoint belongs to:

```
Client                                                                    RD
|----- GET /rd-lookup/gp?ep=node1 ----->
|
|<-- 2.05 Content "</rd>;d=domain1,</rd>;d=domain2 -----
```

Req: GET /rd-lookup/gp?ep=node1

Res: 2.05 Content

<coap://host:port>;gp="lights1";ep="node1",

## 8. New Link-Format Attributes

When using the CoRE Link Format to describe resources being discovered by or posted to a resource directory service, additional information about those resources is useful. This specification defines the following new attributes for use in the CoRE Link Format [RFC6690]:

```
link-extension    = ( "ins" "=" quoted-string ) ; Max 63 bytes
link-extension    = ( "exp" )
```

### 8.1. Resource Instance 'ins' attribute

The Resource Instance "ins" attribute is an identifier for this resource, which makes it possible to distinguish from other similar resources. This attribute is similar in use to the "Instance" portion of a DNS-SD record, and SHOULD be unique across resources with the same Resource Type attribute in the domain it is used. A Resource Instance might be a descriptive string like "Ceiling Light, Room 3", a short ID like "AF39" or a unique UUID or iNumber. This attribute is used by a Resource Directory to distinguish between multiple instances of the same resource type within a system.

This attribute MUST be no more than 63 bytes in length. The resource identifier attribute MUST NOT appear more than once in a link description.

### 8.2. Export 'exp' attribute

The Export "exp" attribute is used as a flag to indicate that a link description MAY be exported by a resource directory to external directories.

The CoRE Link Format is used for many purposes between CoAP endpoints. Some are useful mainly locally, for example checking the observability of a resource before accessing it, determining the size of a resource, or traversing dynamic resource structures. However, other links are very useful to be exported to other directories, for example the entry point resource to a functional service.

## 9. Security Considerations

This document needs the same security considerations as described in Section 7 of [RFC5988] and Section 6 of [RFC6690]. The /.well-known/core resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [I-D.ietf-core-coap].

Access control SHOULD be performed separately for the RD Function Set and the RD Lookup Function Set, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the domain, endpoint or resource level.

## 10. IANA Considerations

"core.rd", "core.rd-group" and "core.rd-lookup" resource types need to be registered with the resource type registry defined by [RFC6690].

The "exp" attribute needs to be registered when a future Web Linking attribute is created.

## 11. Acknowledgments

Szymon Sasin, Kerry Lynn, Esko Dijk, Peter van der Stok, Anders Brandt, Matthieu Vial, Sampo Ukkola and Linyi Tian have provided helpful comments, discussions and ideas to improve and shape this document. The authors would also like to thank their colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

## 12. Changelog

Changes from -04 to -05:

- o Restricted Update to parameter updates.
- o Added pagination support for the Lookup interface.
- o Minor editing, bug fixes and reference updates.
- o Added group support.

- o Changed rt= to et= for the registration & update interface

Changes from -03 to -04:

- o Added the ins= parameter back for the DNS-SD mapping.
- o Integrated the Simple Directory Discovery from Carsten.
- o Editorial improvements.
- o Fixed the use of ETags.

Changes from -02 to -03:

- o Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- o Updated REST interface descriptions to use RFC6570 URI Template format.
- o Introduced an improved RD Lookup design as its own function set.
- o Improved the security considerations section.
- o Made the POST registration interface idempotent by requiring the ep= paramter to be present.

Changes from -01 to -02:

- o Added a terminology section.
- o Changed the including of an ETag in registration or update to a MAY.
- o Added the concept of an RD Domain and a registration parameter for it.
- o Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- o Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

### 13. References

## 13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

## 13.2. Informative References

- [I-D.brandt-coap-subnet-discovery]  
Brandt, A., "Discovery of CoAP servers across subnets", draft-brandt-coap-subnet-discovery-00 (work in progress), March 2011.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.vanderstok-core-bc]  
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.
- [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, June 1999.
- [RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

Authors' Addresses

Zach Shelby  
Sensinode  
Kidekuja 2  
Vuokatti 88600  
FINLAND

Phone: +358407796297  
Email: zach@sensinode.com

Srdjan Krco  
Ericsson

Phone:  
Email: srdjan.krco@ericsson.com

Carsten Bormann  
Universitaet Bremen TZI  
Postfach 330440  
Bremen D-28359  
Germany

Phone: +49-421-218-63921  
Fax: +49-421-218-7000  
Email: cabo@tzi.org



CoRE  
Internet-Draft  
Intended status: Informational  
Expires: January 11, 2013

P. van der Stok, Ed.  
vanderstok consultancy  
K. Lynn  
Consultant  
A. Brandt  
Sigma Designs  
July 10, 2012

CoRE Discovery, Naming, and Addressing  
draft-vanderstok-core-dna-02

Abstract

This is a working document intended to focus discussion and refine draft language for the CoAP protocol specification (or other proposed standards) in the areas of discovery, naming, and addressing. Requirements on discovery are motivated. Special emphasis is placed on group definition and discovery. Examples show how groups can be represented in CoAP Resource Directories or DNS-SD.

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 January 11, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect



to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .                  | 3  |
| 1.1. Terminology . . . . .                 | 3  |
| 1.2. Motivation . . . . .                  | 4  |
| 1.3. Applicability . . . . .               | 4  |
| 2. Architecture . . . . .                  | 5  |
| 3. Use Cases . . . . .                     | 5  |
| 3.1. Discovery scope . . . . .             | 6  |
| 3.2. Interactive mapping . . . . .         | 6  |
| 3.3. M2M mapping . . . . .                 | 7  |
| 3.4. End-point grouping . . . . .          | 7  |
| 3.5. Discovery queries . . . . .           | 10 |
| 3.6. Sleeping devices . . . . .            | 11 |
| 4. DNS and RD examples . . . . .           | 12 |
| 4.1. DNS-SD examples . . . . .             | 13 |
| 4.2. RD examples . . . . .                 | 19 |
| 5. IANA Considerations . . . . .           | 24 |
| 6. Security Considerations . . . . .       | 24 |
| 6.1. DNS Security considerations . . . . . | 24 |
| 6.2. RD Security considerations . . . . .  | 26 |
| 7. Acknowledgments . . . . .               | 26 |
| 8. Changelog . . . . .                     | 26 |
| 9. References . . . . .                    | 26 |
| 9.1. Normative References . . . . .        | 26 |
| 9.2. Informative References . . . . .      | 28 |
| Authors' Addresses . . . . .               | 29 |

## 1. Introduction

The CoRE working group is chartered to design and standardize a Constrained Application Protocol (CoAP) for resource constrained devices and networks [I-D.ietf-core-coap]. The requirements for CoRE are documented in [I-D.shelby-core-coap-req]. This draft discusses the requirements on service discovery for M2M and interactive applications using resource constrained devices. We propose the use of DNS-SD [I-D.cheshire-dnsext-dns-sd] and Resource Directory (RD) [I-D.shelby-core-resource-directory] to satisfy the requirements. The proposal relies heavily on naming and addressing conventions. Special emphasis is placed on the definition, naming, and discovery of groups.

### 1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. Additional privileged words are described below.

A "device" is a physical processor connected to at least one link through a network interface. Each interface has at least one IP unicast address. The IP address is optionally bound to a host name, which may be a Fully Qualified Domain Name (FQDN).

An "end-point" corresponds to a "service" and is identified by a unique {protocol, host, port} tuple. The identity of an endpoint may be specified by the 'scheme' and 'authority' parts of a URI [RFC3986]. 'Protocol' is a label that indicates application and transport protocol bindings as well as default port (if port is not specified) and possibly default semantics such as web-linking [RFC5988]. 'Host' corresponds to the [RFC3986] ABNF production as updated by [I-D.ietf-6man-uri-zoneid].

A "service type", e.g. `_bldg-ctrl._tcp`, is equivalent to the 'protocol' label described above. It identifies an application protocol, typically defined by a Standards Development Organization (SDO), and is ultimately registered with IANA [I-D.cheshire-dnsext-dns-sd]. The SDO may additionally specify service subtypes (e.g. `_light`, `_onoff-control`, `_air-flow` ...) to designate units of functionality, the attributes of the subtypes, and the primitives acting on the attributes.

A "resource" is any attribute of an end-point that can be acted upon by REST methods. A resource is identified by a URI, that is, an end-point plus a 'path' component [RFC3986].

A "Function Set" is a service subtype with a set of resources and behaviors that may be accessed through a REST interface. A Function Set will typically be described by a base URI plus an interface definition as described in [I-D.shelby-core-interfaces]. The interface definition may specify the naming patterns of subordinate resources and the methods that act on them.

A "Profile" is a group of Function Sets defined by a specification.

A "Collection" is a set of two or more homogeneous subordinate resources that may be acted upon in the aggregate by sending messages to their parent resource, or individually by sending messages to the collection member.

A "Device type" describes a standardized set of Function Sets that satisfy a use case for a hosting device.

A "group" is a set of end-points.

A "multicast group" is a group of end-points that share a multicast address. The multicast address is optionally bound to a FQDN that identifies the multicast group. For the purposes of this document, a (multicast) group generally hosts one Function Set.

## 1.2. Motivation

In this draft, we focus and expand discussions on requirements pertaining to discovery, naming, and addressing, including REQ8 of the CoRE charter:

REQ8:    A definition of how to use CoAP to advertise about or query for a Device's description. This description may include the device name and a list of its Resources, each with a URL, an interface description URI (pointing e.g. to a Web Application Description Language (WADL) document) and an optional name or identifier. The name taxonomy used for this description will be consistent with other IETF work, e.g. draft-cheshire-dnsext-dns-sd. [charter]

The basis of this draft originated in [I-D.vanderstok-core-bc].

## 1.3. Applicability

This note applies to discovery of services for commissioning and auto-configuration in building networks (for example: residential and office). The examples are inspired by the building environment. The proposed solutions and recommendations can be applied more generally.

## 2. Architecture

This section illustrates the aspects of naming schemes and their support by DNS-based Service Discovery (DNS-SD) [I-D.cheshire-dnsext-dns-sd], Extended Multicast DNS (xmDNS) [I-D.lynn-homenet-site-mdns], and the Resource Directory (RD) [I-D.shelby-core-resource-directory] on a set of network architectures.

The basic network for low-power nodes can be composed of low-resource nodes sharing the same IPv6 prefix and connected to low-power links like IEEE 802.15.4, ITU-T G.9959, or Powerline. The "lowpan" is a good example of such a network [RFC4944], [RFC6282]. The network can be either isolated or connected. This draft assumes that application profiles are defined above coap or http, for example, applications as specified by the ZigBee Smart Energy Profile 2.0 (SEP2), Obix, or BACnet IT working groups. The naming and discovery solutions presented here are applicable to multiple interconnected subnets. Example network architectures are:

- An isolated lowpan consists of at least two lowpan devices, one of which is an edge router that is not connected to a backbone. A Resource Directory may be situated on the edge router. Alternatively, xmDNS responders may execute on each device.
- A connected lowpan consists of at least two lowpan devices, one of which is an edge router that is connected to a backbone. A Resource Directory may be situated on an edge router. Alternatively, xmDNS responders may execute on each device or the DNS-SD service may be available over the backbone.
- Interconnected lowpans consist of at least two lowpans connected via edge routers to the same backbone. A Resource Directory may be situated on each edge router. Alternatively, xmDNS responders may execute on each device or the DNS-SD service may be available over the backbone.
- A site is a set of interconnected subnets that is locally administered. A site may include zero or more lowpans. Border routers may prevent some messages from passing into or out of the site.

In certain scenarios, the domain may correspond to the network topology. In the general case, the domain and network subnet structure may differ.

## 3. Use Cases

The use of service discovery is presented in two environments: (1) interactive service discovery, and (2) M2M service discovery. From

the use cases we derive the types of queries that service discovery should support. In particular, a primary motivation is the discovery of groups that support a given Function Set.

### 3.1. Discovery scope

In the simplest case the discovery scope is limited to the services and resources provided on the LowPAN. In more complex cases, discovery has a scope that can be defined with domain names. The authority part of a URI [RFC3986] can express the location of the device hosting the service. A common example is the naming associated with the structure of a building. A device may acquire a FQDN that relates directly to its location in the building. For example, power-strip.office4.floor1.example.com (or shorter ps.o4.fl.example.com) refers to a power-strip with device name "power-strip" (or short "ps") in office4 at floor1 in the building of the company example.com. Another naming scheme can be functional like TV1.media.IT.example.com, possibly referring to TV number 1 maintained by the media group of the IT department of the example organisation. Domain naming can be used to express that devices are situated in the same building area or belong to the same organisational units. Multiple FQDNs can identify a given device.

The DNS provides the mapping from Fully Qualified Domain Name (FQDN) to network address and vice-versa. The RD realises domain names to end-points. The binding of domain names to a physical device (for example, assigning a given FQDN to the TV in the corner) depends on the operational conditions as described below.

### 3.2. Interactive mapping

In the residential context, naming of the device is done by the occupant of the home. After connecting the device to the network, an IP-address is assigned, possibly based on the EUI-64 value of the network interface. The occupant can use a remote control with a graphical user interface to display all devices that provide a given service (e.g. a "lighting" service). The remote control prompts the occupant to identify the (default named) devices, possibly accompanied by on/off switching, barcode scanning, or other manual intervention. The occupant can provide a meaningful name that will be bound to that device. The installation steps can be as follows: Service discovery returns all interfaces on which the specified service is available. The occupant, with or without additional physical support, establishes the binding between an IP address (based on MAC address or other unique identifier) and the name of the device providing the service, plus its relationship with other devices or function sets in the system. In some cases a device has multiple names, and an additional mapping between user specified name

and an automatically generated DNS name is supported.

### 3.3. M2M mapping

In the commercial context (e.g. office buildings) it is usual to employ a Commissioning Tool (CT) to provide the mapping from physical device to IP network address or FQDN. The professional context is more rigid than the home because the absence of devices and also the unwanted presence of devices needs to be detected. Devices are named by an architect or installation contractor. Names can be generated automatically and need not be human-friendly. The CT contains the names and the physical locations of the devices. At commissioning time the interfaces have acquired a network address, possibly based on the EUI-64. The physical device is identified by reading in a unique identifier (e.g. EUI-64 of interface, UUID of device) with a reader (e.g. barcode reader). Consequently, the device name to network address binding is stored into DNS (or elsewhere). Alternatives for identifying devices are pushing buttons on the device or remotely switching on/off the device.

### 3.4. End-point grouping

Groups can be used to express that end-points are related by supporting a specific Function Set (e.g. HVAC equipment controlled by the closest temperature sensor). Grouping is also necessary when a set of end-points has to react together, more or less synchronously, to a sequence of commands sent by one or more devices. A common example is provided by lighting applications where a subset of lights in the building are dimmed to the same level, set to the same colour, or switched off simultaneously. Another example is provided by a power-strip supporting a set of power-outlets. Power-outlets are switched on/off individually or all together. Other examples concern the home, such as a "sleep mode" setting of all media devices in the home when the user activates the night scene.

Group naming is done the same way as for device naming. Related end-points are grouped and named. The group name is constructed like a FQDN with the group name as prefix. Addressing the group can be done in two ways: (1) by addressing each end-point of the group individually (which requires serial access), or (2) by defining a multicast address for a multicast group. In the latter case, each hosting device must enable reception by a given end-point of the messages sent to the multicast address. The members of the multicast group must have identical port number and path, because the requests are specified in a single multicast message.

The Function Set specified in a message can contain a collection of resources of the same subtype. The Function Set path postfixed with

an identifier refers to the individual resources of the collection. For example: the /path/light points to the resource collection of the service subtype light. Each member of the collection can be identified with /path/light/x, with x in {1,2,3,...}. Consequently, the /path/light/1/onoff specifies the onoff resource of collection member 1 of Function Set with /path/light, and /path/light/onoff specifies the resource onoff of all collection members contained by the Function Set with /path/light. When /path/light/onoff is used in a multicast message, it is interpreted as a message to a single light resource by devices having only one, and to all members of the collection for devices having several light resources.

In [I-D.shelby-core-interfaces] the Batch interface is used to manipulate a collection of subresources with one message. Both the batch and the collection are static. The collection contains homogeneous resources where the batch can contain heterogeneous resources.

It is expected that SDOs will standardize resource types, profiles (path names) and group naming conventions. Manufacturers design the functions supported by a device and select the Profiles, resources and collections. Each manufacturer can precede the profile path names with a manufacturer defined path prefix; for example when a device supports multiple standards. Domain name, device name, group name and group members are installation dependent.

Figure 1 illustrates some of the concepts described above. A device is identified with the name "Power-strip". In the example, no domain names are associated with the device, and the name "power-strip" resolves to a Unique Local Address (ULA)[RFC4193]. The device provides two end-points: one delivers a http service and the second a CoAP service. The http server and CoAP server share the same IP-address and use different ports 80 and 61616 respectively. Two different service subtypes, one identified by Function Set, ps, and one identified by Function Set, pm, are supported by one end-point. The Function Set "Power strip" contains a collection of four resources, "Outlet 1" to "Outlet 4", each one accessible via the accompanying paths /ps/1 to ps/4. The path /ps interfaces to the entire resource collection. The attribute "output" is defined in the service subtype specification.

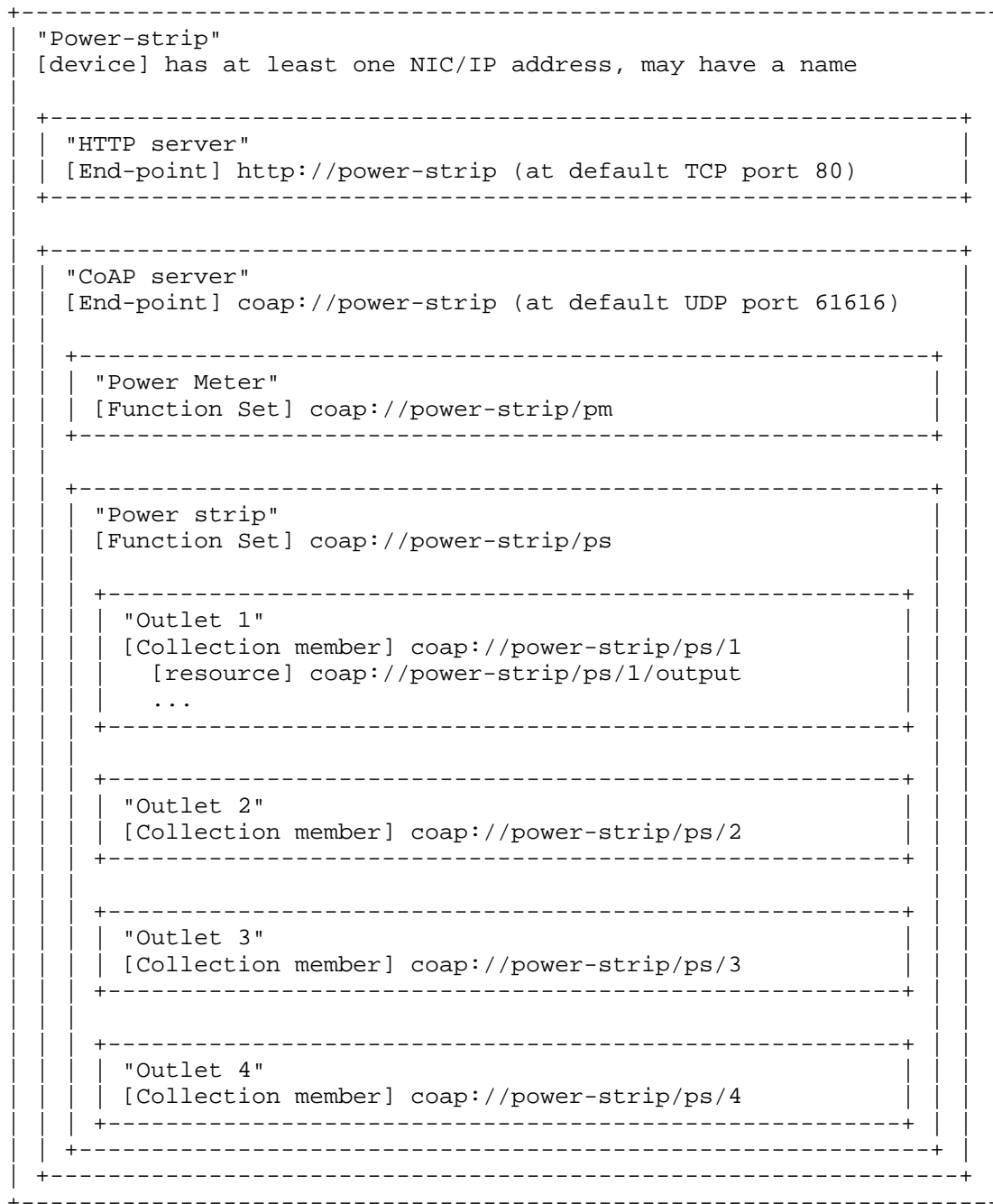


Figure 1: device with end-points, Function Sets and resources



### 3.5. Discovery queries

The following conditions are assumed to be valid. A device has acquired an IP address and a name, possibly stored in DNS. The resource naming (path) in the device obeys a standardized profile. The resource types of the Function Sets are also standardized. A device may belong to a domain.

Service discovery should support that a device can learn its domain(s) and all the end-points within a domain providing a given service (e.g. temperature measurement). The device learns of these end-points the path name that prefixes the path names defined by the profile. Devices need to learn the groups to which they belong and learn all the members of those groups. This section motivates that a discovery service supports the following queries:

| Goal            | Description   |
|-----------------|---|
| Name_resolution | Resolve the group or device name to IP address and optional port number                           |
| Return_server   | Return all end-points (Function-Sets) supporting a given service (sub-)type within a given domain |
| Create_group    | Create a group of end-points providing a given service (sub-)type within a given domain           |
| Enroll_member   | Enroll an end-point as member of a given group  |
| Remove_member   | Remove an end-point as member of a given group  |
| Return_group    | Return all groups of which a given end-point (device) is a member                                 |
| Return_member   | Return end-points belonging to a given group  |

Name\_resolution is supported by DNS and CoAP resource discovery. Names are required in the context of home control and manual setup of installations. Names are persistent and meaningful as compared to IP addresses and are preferably used in applications when IP addresses can change.

Return\_server is the most common use of service discovery and was originally designed for interactive use. The canonical IT example is finding all printers within a zone, which allows a user to select the desired printer from the returned list. Another example is in the context of UPnP [UPNP], where all media players are returned on a screen and the user can select the desired media player on the screen and play the selected content. In M2M applications, the returned names are not displayed on a screen but an application uses the returned list to select a (set of) Function Set(s) to control. Consequently, names in M2M applications need not be human interpretable (for example, they can be unique numbers).

Create\_group is useful in commissioning scenarios, where end-points

need to be grouped to receive the same command in a possibly synchronous fashion. Groups can also be created to express relations between devices such as ownership. The command creates a group name and creates a list of the members of the group. When the group is a multicast group, the command defines a unique multicast address and port, and specifies the path.

Enroll\_member supports network and device reconfiguration. When the physical lay-out of an installation changes because devices are added, changed or removed, the associated groups also need to be modified.

Remove\_member, see motivation under Enroll\_member

Return\_group is needed to learn the groups of which an end-point is member. The command is necessary for commissioning purposes where a Commissioning Tool (CT) is used. The CT, on the basis of designs provided by architects, decorators, sound/light engineers, defines groups and group members and stores that information in the service discovery database. In the next phase, the members of a group may need to learn their membership from the service discovery to enable reception of multicast messages.

Return\_member can be used to learn which end-points are member of a given group. This command is useful in connection with Return\_group. The end-point knowing to which groups it belongs can establish communication with the group members. For example, membership of a group instructs new devices, replacing faulty ones, which other devices share access rights or need to be consulted regularly.

### 3.6. Sleeping devices

This section suggests that service discovery of sleeping devices is mostly a matter of discovering the proxy. It is expected that a proxy will handle communications for the sleeping device, as expressed in [I-D.giacomin-core-sleepy-option] and in [I-D.vial-core-mirror-proxy]. The message sent to the sleeping device is directed to the proxy. The proxy will send the message on with a delay, or send the result of a function on the history of messages, when the sleeping device is ready. Different communication protocols between proxy and sleeping device are described in [I-D.giacomin-core-sleepy-option] and in [I-D.vial-core-mirror-proxy]. The setting up of the proxy is preferably standardized for a large set of proxy types. During the setting-up process, (offline or online) the proxy will take over all the entry-points of the sleeping device. The entry-points of the proxy can be entered into the discovery repository and consequently discovered like any other device.

For groups, two cases need to be considered (1) sleeping device is member of a group and receives group messages, and (2) the sleeping device sends messages to a group. Ad (1), when the sleeping device needs to receive messages sent to a group, the proxy will receive those messages and the end-point of the proxy is entered as group member to receive the group messages. Ad (2) When the sleeping device sends messages to a group, it is preferable that the sleeping device sends just one multicast message to the group to minimize energy costs. It is required that when one member of the group receives the message, all other group members receive it as well (unanimity), covered by the "reliability" REQ1 in [I-D.ietf-core-groupcomm]). A simple broadcast over a lowpan will not always succeed and additional multicast algorithms like Trickle [RFC6206] need to be introduced.

#### 4. DNS and RD examples

The following device configuration and environment are assumed for the examples. The devices are placed on floor-x (fx) in two rooms room-y (ry) and room-z (rz). Both rooms contain a powerstrip with a powermeter and four power outlets. In each room there are two luminaires and one presence sensor (PIR). Each luminaire contains a dimmable light and a light sensor. Per floor there is a clock to set day and night time modes of the devices. The domains are: ry.fx.bldg.org, rz.fx.bldg.org and fx.bldg.org. The device names of the 4 luminaires are lm00203, lm00204, lm00205, and lm00206. The device names of the two powerstrips are ps0057, and ps0078. The paths of the Function Sets of the luminaire are: /lamp with resource /lamp/dim for the dimmable light, and /light with resource /light/lumen for the light sensor. The Function Set path for four outlets is /ps with resource /ps/output. The path of each individual outlet is /ps/x with x in {1,2,3,4}, and with resource ps/x/output. The names of the two PIRs are pir0 and pir1. The Function Set path of the PIR is /occup, also being the resource.

Relating location to the domain name is a relevant example of domain naming. Multiple domain names, related to other application aspects, can be specified and applied simultaneously.

As described in section 3, devices do not announce themselves to the discovery repository, as is usual for IT applications, but they are entered (partially) with the aid of a central tool, for example a Remote Control, dedicated device, IPAD or other means.

The examples are constructed such that DNS can be used as repository for the RD.

#### 4.1. DNS-SD examples

##### 4.1.1. Basic Concepts

In conformance with [I-D.cheshire-dnsext-dns-sd], DNS-based discovery uses A or AAAA, PTR, SRV, and TXT Resource Records (RR). The SRV RR [RFC2782] specifies an end-point. An associated (identically named) TXT RR can contain a URI path. The SRV/TXT pair can specify a Function Set. An A or AAAA RR [RFC1035] binds a device name or multicast group name to an IP address. The PTR RR binds a service type to an end-point or Function Sets.

In cases where the end-point port may be dynamic, e.g. in the IPHC [RFC6282] compressible range, a new 'coap+srv' scheme is proposed (after [I-D.jennings-http-srv]). The authority part of a coap+srv URI specifies the name and location of an SRV record, which in turn contains values for device (IP address) and port.

##### 4.1.2. Commissioning devices

Commissioning is the process to store the relation between a FQDN and a device. It is assumed that either a Remote Control (RC) in the home or a Commissioning Tool (CT) in the professional domain store the relation in DNS.

In the professional domain, the CT is assumed to contain information about the devices as prescribed by architect or installation company. The information in the CT contains device name, device domain name, and location in the building, but the relation with the installed device, identified with a unique identifier (e.g. EUI-64) is not established. By reading a bar code (or pushing buttons, switching on/off equipment, etc.), the CT learns the unique identifier of the device to be commissioned. All kinds of techniques can be used to establish the relation between IP address and unique identifier. When the identifier is the EUI-64 value, the IP address of the device can be constructed. When the identifier is not the EUI-64, a proprietary protocol can be used to ask a given device its identifier. Etc. etc. The CT can learn the end-points (services) available on the device by querying /.well-known/core. In some cases the CT already obtained the end-points from a configuration file. Given these data, the CT can enter the devices and its services into DNS. Either automatically, or on instructions of an operator, the CT defines the groups in the DNS.

The home domain is different from the professional domain in the sense that no configuration information exists. The RC can for example use xmDNS to learn the addresses of all the devices present in its site. The RC can query devices for the presence of a given

service. The RC can query DNS for its own domain name and use that for the other devices in the site. Once (new) devices are named, this information can be stored in DNS for use in the network.

#### 4.1.3. device examples

The relation between device name and IP address is expressed for the example devices in the following table.

|                         |                    |
|-------------------------|--------------------|
| lm00203.ry.fx.bldg.org. | IN AAAA fdfd::1234 |
| lm00204.ry.fx.bldg.org. | IN AAAA fdfd::1235 |
| ps0057.ry.fx.bldg.org.  | IN AAAA fdfd::1236 |
| pir1.ry.fx.bldg.org.    | IN AAAA fdfd::1237 |
| lm00205.rz.fx.bldg.org. | IN AAAA fdfd::1238 |
| lm00206.rz.fx.bldg.org. | IN AAAA fdfd::1239 |
| ps0058.rz.fx.bldg.org.  | IN AAAA fdfd::1240 |
| pir2.rz.fx.bldg.org.    | IN AAAA fdfd::1241 |
| clock.fx.bldg.org.      | IN AAAA fdfd::1242 |

The next part defines the Resource Records to specify the end-points and their Function Sets. The names of the SRV RRs need to be unique to the DNS server, and follow the naming suggested in [I-D.cheshire-dnsext-dns-sd]. An end-point is represented with one SRV RR with a name "\_service.domain". A Function-Set is represented with a SRV/TXT pair with name "subtype.\_sub.\_service.domain".

The service type "\_bc.\_udp" is assumed to exist with the service subtypes "lamp", "sensor", "power", "presence", and "timer". Unique names of the SRV RRs can be created from the service subtype prefixed by the EUI-64 value. With EUI-64 value "1234" the SRV name 1234\_bc.\_udp.domain can be created for the corresponding end-point. Given that the service \_bc.\_udp supports subtype lamp the name 1234lamp.\_sub.\_bc.\_udp.domain is created for for each SRV/TXT pair describing a Function-Set serving the lamp subtype. They are valid within the authority zone, bldg.org, of the name server. For presentation purposes, 1234\_bc is short for 1234\_bc.\_udp.bldg.org, 1234lamp is short for 1234lamp.\_sub.\_bc.\_udp.bldg.org, etc. The luminaires with name "lm00xxx" provide each one end-point hosting two Function Sets with the paths /lamp and /light.

| end-point |                  | host name              |
|-----------|------------------|------------------------|
| 1234_bc   | IN SRV 0 0 Port  | lm00203.ry.fx.bldg.org |
|           | IN TXT txtvers=1 |                        |
| 2345_bc   | IN SRV 0 0 Port  | lm00204.ry.fx.bldg.org |
|           | IN TXT txtvers=1 |                        |
| 3456_bc   | IN SRV 0 0 Port  | lm00205.rz.fx.bldg.org |
|           | IN TXT txtvers=1 |                        |
| 4567_bc   | IN SRV 0 0 Port  | lm00206.rz.fx.bldg.org |
|           | IN TXT txtvers=1 |                        |
| 5678_bc   | IN SRV 0 0 Port  | ps0057.ry.fx.bldg.org  |
|           | IN TXT txtvers=1 |                        |
| 6789_bc   | IN SRV 0 0 Port  | ps0058.rz.fx.bldg.org  |
|           | IN TXT txtvers=1 |                        |
| 7890_bc   | IN SRV 0 0 Port  | pir1.ry.fx.bldg.org    |
|           | IN TXT txtvers=1 |                        |
| 8901_bc   | IN SRV 0 0 Port  | pir2.rz.fx.bldg.org    |
|           | IN TXT txtvers=1 |                        |
| 9012_bc   | IN SRV 0 0 Port  | clock.fx.bldg.org      |
|           | IN TXT txtvers=1 |                        |

The above list of SRV RRs specifies the attributes of the end-points: devices, port numbers, and IP addresses with related AAAA RR. The Function Sets are specified with a another set of SRV/TXT pairs. The accompanying TXT RR specify the paths of the associated Function Set(s). The accompanying example table is:

| Function Set |                              | host name              |
|--------------|------------------------------|------------------------|
| 1234lamp     | IN SRV 0 0 Port              | lm00203.ry.fx.bldg.org |
|              | IN TXT txtvers=1 path=/lamp  |                        |
| 1234sensor   | IN SRV 0 0 Port              | lm00203.ry.fx.bldg.org |
|              | IN TXT txtvers=1 path=/light |                        |
| 2345lamp     | IN SRV 0 0 Port              | lm00204.ry.fx.bldg.org |
|              | IN TXT txtvers=1 path=/lamp  |                        |
| 2345sensor   | IN SRV 0 0 Port              | lm00204.ry.fx.bldg.org |
|              | IN TXT txtvers=1 path=/light |                        |
| 5678power    | IN SRV 0 0 Port              | ps0057.ry.fx.bldg.org  |
|              | IN TXT txtvers=1 path=/ps    |                        |
| 7890presence | IN SRV 0 0 Port              | pir1.ry.fx.bldg.org    |
|              | IN TXT txtvers=1 path=/occup |                        |
| 3456lamp     | IN SRV 0 0 Port              | lm00205.rz.fx.bldg.org |
|              | IN TXT txtvers=1 path=/lamp  |                        |
| 3456sensor   | IN SRV 0 0 Port              | lm00205.rz.fx.bldg.org |
|              | IN TXT txtvers=1 path=/light |                        |
| 4567lamp     | IN SRV 0 0 Port              | lm00206.rz.fx.bldg.org |
|              | IN TXT txtvers=1 path=/lamp  |                        |
| 4567sensor   | IN SRV 0 0 Port              | lm00206.rz.fx.bldg.org |
|              | IN TXT txtvers=1 path=/light |                        |
| 6789power    | IN SRV 0 0 Port              | ps0058.rz.fx.bldg.org  |
|              | IN TXT txtvers=1 path=/ps    |                        |
| 8901presence | IN SRV 0 0 Port              | pir2.rz.fx.bldg.org    |
|              | IN TXT txtvers=1 path=/occup |                        |
| 9012timer    | IN SRV 0 0 Port              | clock.fx.bldg.org      |
|              | IN TXT txtvers=1 path=/time  |                        |

The names of the SRV records can be created automatically, as long as they identify the SRV records uniquely within the set of RR entries in the DNS zone. The SRV record with name xxxxpwr stands for a power collection, accessed via the path /ps which refers to a Function Set that contains a collection of two or more resources.

PTR records specify the possible service discovery queries. The names of the PTR records are the names of the service (sub)types, defined by IANA, and their values refer to the names of the associated end-points (SRV records) or Function-Sets (SRV/TXT records). To support the query "all lamps within fx.bldg.org", the following PTR records need to be added for service subtype: lamp.\_sub.\_bc.\_udp.

```

lamp._sub._bc._udp.bldg.org IN PTR 1234lamp
lamp._sub._bc._udp.bldg.org IN PTR 2345lamp
lamp._sub._bc._udp.bldg.org IN PTR 3456lamp
lamp._sub._bc._udp.bldg.org IN PTR 4567lamp

```

Where xxxxlamp is still short for xxxxlamp.\_sub.\_bc.\_udp.bldg.org.

Equally to query all services within a domain, PTR records with as name the building control service, "\_bc.udp", refer to all SRV records describing all discoverable end-points.

```
_bc._udp.bldg.org IN PTR 1234_bc._udp.bldg.org
_bc._udp.bldg.org IN PTR 2345_bc._udp.bldg.org
_bc._udp.bldg.org IN PTR 3456_bc._udp.bldg.org
_bc._udp.bldg.org IN PTR 4567_bc._udp.bldg.org
_bc._udp.bldg.org IN PTR 5678_bc._udp.bldg.org
_bc._udp.bldg.org IN PTR etc, etc.
```

It is shown above how PTR records support the queries filtered on service type. Filtering on domain can be done adding additional PTR records which select the Function Sets of a given type within a given domain. The set of PTR records below filters on all lamps within domain ry.fx.bldg.org.

```
lamp._sub._bc._udp.ry.fx.bldg.org. IN PTR 1234lamp
lamp._sub._bc._udp.ry.fx.bldg.org. IN PTR 2345lamp
```

#### 4.1.4. Group examples

As an example, five multicast-groups are defined to group all lamps on floor "fx", all lamps in office "ry", all lamps in office "rz", all power-strips on floor "fx", and all devices in the building controlled by a central timer. The multicast-group names are entered into DNS like the device names to enable resolution from multicast-group name to multicast address.

```
lamp-fx.fx.bldg.org.      IN AAAA ff15::11
lamp-ry.ry.fx.bldg.org.   IN AAAA ff15::12
lamp-rz.rz.fx.bldg.org.   IN AAAA ff15::13
power-fx.fx.bldg.org.     IN AAAA ff15::14
timer-bldg.bldg.org.      IN AAAA ff15::15
```

It is expected that SDOs will specify naming conventions for group names, extending the service (sub)type names for end-points and Function Sets.

The path and port of the multicast-groups is defined with SRV and TXT RRs. Accordingly the group is bound to end-points (SRV RR) and Function Sets (SRV+TXT RR). For convenience we assume that a multicast group is bound to a service subtype. In the example below the groups concern the service subtypes "lamp", "power", and "timer". (Remark gpl-lamp is short for gpl-lamp.\_sub.\_bc.\_udp.bldg.org, etc.)



```
gp1-lamp IN SRV 0 0 Port    lamp-fx.fx.bldg.org.  
          IN TXT path=/lamp  
gp2-lamp IN SRV 0 0 Port    lamp-ry.ry.fx.bldg.org.  
          IN TXT path=/lamp  
gp3-lamp IN SRV 0 0 Port    lamp-rz.rz.fx.bldg.org.  
          IN TXT path=/lamp  
gp-power IN SRV 0 0 Port    power-fx.fx.bldg.org.  
          IN TXT path=/ps  
gp-timer IN SRV 0 0 Port    timer-bldg.bldg.org.  
          IN TXT path=/time
```

The groups for the power strips need extra attention because the power strips include a collection of resources. The path to the group can be defined as /ps or as /ps/x with x in {1,2,3,4}. When using /ps/x the group contains the outlet x of the power strips. Using the path /ps as done in the table above refers to all outlets of a powerstrip. When sending a message to the power-fx group with as path /ps/x then the message will be received by Function Sets with path ps/x only.

The members of the groups can be stored in DNS by using the reverse DNS resolution technique. It is not unusual that a given IP address refers to multiple FQDNs. Extrapolating to group names extends the reverse DNS resolution in a natural manner. Below the members of group lamp-fx.fx.bldg.org with IP address ff15::11 containing all four lamps is shown.

```
1.1.0.....0.5.1.f.f.IP6.arpa. IN PTR lm00206.rz.fx.bldg.org.  
1.1.0.....0.5.1.f.f.IP6.arpa. IN PTR lm00205.rz.fx.bldg.org.  
1.1.0.....0.5.1.f.f.IP6.arpa. IN PTR lm00204.ry.fx.bldg.org.  
1.1.0.....0.5.1.f.f.IP6.arpa. IN PTR lm00203.ry.fx.bldg.org.  
1.1.0.....0.5.1.f.f.IP6.arpa. IN PTR lamp-fx.fx.bldg.org.
```

With the above table queries like "return all members of lamp-fx" can be answered.

Additional tables are needed to specify the multicast groups to which the end-point of a device belongs. A PTR RR with the name of the Function Set can refer to the name of the group. In the table below the Function Set "1234lamp" is member of the groups "lamp-fx" and "lamp-ry":

```
1234lamp IN PTR lamp-fx.fx.bldg.org  
1234lamp IN PTR lamp-ry.ry.fx.bldg.org
```

Consequently, a device can query DNS for the groups to which its Function Set belongs, and consecutively enable the reception of the associated multicast messages.

#### 4.1.5. Discovery validation

This section describes how the discovery requirements are met with DNS-SD. It is assumed that the DNS server tables are correctly filled in.

**Name\_resolution:** Group- or device-name is resolved to IP address by sending a query to DNS to return all AAAA records with the given name.

**Return\_server:** Suppose the given service is defined by instance.\_sub.\_service.domain. All Function Sets supporting this service in the domain are found by sending a query to DNS to return all PTR records with the name instance.\_sub.\_service.domain. The returned PTR records refer to the names of the SRV/TXT pairs describing the port, FQDN and path of the associated Function Sets. Additional queries return all SRV and TXT records with the names returned by the first query. To query the end-points a query is sent to return all PTR records with name service.domain.

**Create-Group:** Groups are created by creating resource records in DNS as described in section 4.1.4. The filing of the DNS server tables is done by a trusted Remote Control or commissioning Device (see section 4.1.2). Section 6.1 discusses the security aspects.

**Enroll-member, Remove-member:** See Create-Group.

**Return-Group:** Suppose the given Function Set is given by the name instance.\_sub.\_bc.\_service.domain. All groups to which this Function Set belongs are found by sending a query to DNS to return all PTR records with the name instance.\_sub.\_bc.\_service.domain. The returned PTR records refer to the names of the associated groups. Name resolution provides the IP address.

**Return-member:** The members of a group with name group group.domain are found by resolving the group name to an IP address. The members of the group are obtained by sending a query to DNS to return all PTR records with as the name the inverted IP-address. The PRT records refer to the names of the associated devices.

#### 4.2. RD examples

Resource discovery in CoAP handles resource paths (called links) for the resources hosted on the server, augmented with attributes of these resources. A well-known path `"/.well-known/core"` [RFC5785] is a default Function Set for requesting the list of links on a given server [I-D.shelby-core-resource-directory]. The Resource Directory (RD) stores links to resources hosted by other servers. The link-

format [I-D.ietf-core-link-format] defines link extensions to specify the service type and end-point as used by DNS-SD. When querying the Resource Directory for links, filters can be applied to return only links with specified attribute values. A node learns the IP-address of the RD by for example sending a multicast request to a predefined multicast address registered with IANA, or by assuming that the RD is located in the edge router.

Contrary to DNS-SD, the RD has not defined a process which permits SDOs to specify service (sub)types. Consequently, the same service type examples are used for DNS-SD as for RD, where service type postfixed with subtype (DNS) equals "resource type" (RD). The "host" name of DNS is used as "end-point" name for RD.

This draft adheres to the mapping between DNS and link-format, described in [I-D.lynn-core-discovery-mapping].

#### 4.2.1. Commissioning devices

It is assumed that either a Remote Control (RC) in the home or a Commissioning Tool (CT) in the professional domain is used to fill in the RD. Devices cannot enter links into the RD contrary to the suggestion in [I-D.shelby-core-resource-directory]. Both the RC or the CT have to fill in the RD, because at link creation the RD generates a location of the link-entry (e.g. /ed/453), that is returned to the creator of the link. Consequently, the CT or the RD need to create the link, because they need the location to update the link with additional information.

In the professional domain, the CT learns the identity of the device as described earlier, reads the services of the devices with a GET to /.well-known/core on the device, and stores them into the RD.

In the home domain, the RC can read in the EUI-64 of the device to be entered. The user types in domain names and device names on the RC. Consecutively, the RC follows the same procedure as the CT.

#### 4.2.2. Device examples

For convenience, it is assumed that DNS contains the mapping from FQDN to IP address with AAAA RRs and vice-versa with PTR RRs. In all examples the FQDN is used and not the IP-address. It is assumed that the service type "\_bc" has the subtypes: "lamp", "sensor", "power", "presence", and "timer". Registration is done with the following statements by CT or RC to the RD with authority: //rd.example.com/ and path /rd. Each POST command to the RD defines an end-point in the URI and defines the corresponding Function Sets in the payload.

```
POST coap://rd.example.com/rd?ep=lm00203;d=ry.fx.bldg.org
Etag: 0x21
Payload:
</lamp>;rt="lamp._sub._bc",
</light>;rt="sensor._sub._bc"
Res: 2.01 created
Location: /rd/1234
```

The other end-points are defined with (leaving out the response):

```
POST coap://rd.example.com/rd?ep=lm00204;d=ry.fx.bldg.org
Payload:
</lamp>;rt="lamp._sub._bc",
</light>;rt="sensor._sub._bc"

POST coap://rd.example.com/rd?ep=lm00205;d=rz.fx.bldg.org
Payload:
</lamp>;rt="lamp._sub._bc",
</light>;rt="sensor._sub._bc"

POST coap://rd.example.com/rd?ep=lm00206;d=rz.fx.bldg.org
Payload:
</lamp>;rt="lamp._sub._bc",
</light>;rt="sensor._sub._bc"
```

```
POST coap://rd.example.com/rd?ep=ps0057;d=ry.fx.bldg.org
Payload:
```

```
</ps>irt="power._sub._bc",
</ps/1>irt="power._sub._bc",
</ps/2>irt="power._sub._bc",
</ps/3>irt="power._sub._bc",
</ps/4>irt="power._sub._bc"
```

```
POST coap://rd.example.com/rd?ep=ps0058;d=rz.fx.bldg.org
Payload:
```

```
</ps>irt="power._sub._bc",
</ps/1>irt="power._sub._bc",
</ps/2>irt="power._sub._bc",
</ps/3>irt="power._sub._bc",
</ps/4>irt="power._sub._bc"
```

```
POST coap://rd.example.com/rd?ep=pir1;d=ry.fx.bldg.org
Payload:
```

```
</occup>irt="presence._sub._bc"
```

```
POST coap://rd.example.com/rd?ep=pir2;d=rz.fx.bldg.org
Payload:
```

```
</occup>irt="presence._sub._bc"
```

```
POST coap://rd.example.com/rd?ep=clock;d=fx.bldg.org
Payload:
```

```
</time>irt="timer._sub._bc"
```

Update or removal of the groups is done by using the returned location (not shown above) as described in [I-D.shelby-core-resource-directory] for the end-points.

#### 4.2.3. Group examples

The same five multicast groups of the DNS example are used. The group name to address mapping is specified in DNS with AAAA RRs. The group name is not easily identified with an end-point. Therefore the mnemonic gp is added as link-format attribute. The group members are included by citing the end-point names, which allows a unique identification of the members. In all examples the group name is used and not the IP-address. Registration is done with the following statements by CT or RC to the RD with authority: /rd.example.com/ and path /rd, leaving out Etag:, Res:, and Location: lines. The group name is defined in the URI, while all the members (end-points) are specified in the payload. The path in the payload defines together with the end-point the Function Set.

```
POST coap://rd.example.com/rd?gp=lamp-fx;d=fx.bldg.org
Payload:
</lamp>;rt="lamp._sub._bc";ep=lm00206,
</lamp>;rt="lamp._sub._bc";ep=lm00205,
</lamp>;rt="lamp._sub._bc";ep=lm00204,
</lamp>;rt="lamp._sub._bc";ep=lm00203

POST coap://rd.example.com/rd?gp=lamp-ry;d=ry.fx.bldg.org
Payload:
</lamp>;rt="lamp._sub._bc";ep=lm00204,
</lamp>;rt="lamp._sub._bc";ep=lm00203

POST coap://rd.example.com/rd?gp=lamp-rz;d=rz.fx.bldg.org
Payload:
</lamp>;rt="lamp._sub._bc";ep=lm00206,
</lamp>;rt="lamp._sub._bc";ep=lm00205

POST coap://rd.example.com/rd?gp=power-fx;d=fx.bldg.org
Payload:
</ps>;rt="power._sub._bc";ep=ps0057,
</ps>;rt="power._sub._bc";ep=ps0058

POST coap://rd.example.com/rd?gp=timer-bldg;d=bldg.org
Payload:
</time>;rt="timer._sub._bc";ep=clock
```

With the above statements the groups are defined in the RD.

#### 4.2.4. Discovery validation

This section describes how the discovery requirements are met with the RD. It is assumed that the RD tables are correctly filled in.

**Name\_resolution:** When LoWPAN is connected to backbone, it is assumed that DNS is used for name resolution. When LoWPAN is stand-alone without DNS server, IP addresses are used to connect to an interface.

**Return\_server:** Suppose the given service is defined by instance.\_sub.\_service. The query

```
GET coap://rd.example.com/rd-lookup/res?rt=instance.
                                     _sub._service
Res: 2.05 content
<coap://name.domain/path>;rt="instance._sub._service"
```

returns all Function Sets providing the specified service. By changing the "res?" part in the query by "ep?" all end-points are returned.

Create-Group: Groups are created by sending POST commands to RD as described in section 4.2.2. The filling of the RD server tables is done by a trusted Remote Control or commissioning Device (see section 4.2.1). Section 6.1 discusses the security aspects.

Enroll-member, Remove-member: See Create-Group.

Return-Group: Suppose the given end-point is given by the name ep.domain. All groups to which this end-point belongs are found by sending a query

```
GET coap://rd.example.com/rd-lookup/gp?ep=ep.domain
Res: 2.05 content
<coap://name.domain/path>;gp="group.domain"
```

returns all groups of which the specified end-point, ep.domain, is a member.

Return-member: The members of a group with name group group.domain are found by the query

```
GET coap://rd.example.com/rd-lookup/ep?gp=group.domain
Res: 2.05 content
<coap://name.domain/path>;ep="ep.domain"
```

which returns all end-points which are member of the group gp.domain.

## 5. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

## 6. Security Considerations

Security considerations apply to use of DNS and RD separately, because they use different security mechanisms.

### 6.1. DNS Security considerations

Security extensions for DNS are provided by Domain Name System Security Extensions (DNSSEC) as described in [RFC4033], [RFC4034], and [RFC4035]. In particular [RFC4033] describes all documents relevant to DNSSEC. The central design decision of DNSSEC is to provide origin authentication and integrity protection for DNS data.

All transported DNS data are freely accessible. Consequently, all correctly functioning nodes will receive the domain and group names to which they belong, and will receive the addresses of the multicast and unicast destinations, and the multicast address of the groups to which they belong. Accordingly, commands will be sent to the intended destinations and accepted by the intended destinations. All resolvers MUST be configured with a security anchor. The anchor can be stored non-volatile memory or be provided by other out-of-band means.

Updating of DNS data, e.g., zone data MUST be done using the secure dynamic updates mechanism. For example, a commissioning device is used to fill in the DNS server. Both the device sending the update requests (i.e., the commissioning device) and the DNS Server MUST share a Transaction Signature (TSIG) key [RFC2845]. The TSIG key is a symmetric-key and it can be generated using the `dnssec-keygen` primitive. The TSIG key is then used to sign the DNS update message, thus ensuring the authenticity of the request. An access control list can be defined in the DNS to authorize updates to the DNS data. The Berkeley Internet Name Daemon (BIND) implementation of the Internet System Consortium (ISC) provides a mechanism to specify access control lists, ensuring that updates of DNS zone data can only be performed by authorized parties. The `allow-update` option in a zone statement is used to control access to a zone. This option grants clients with a particular TSIG key the permission to update any record of any name in the zone. To allow for finer-granular access control, the `update-policy` option can be used within a zone statement, it specifies a set of rules where each rule either grants or denies permissions to one or more names to be updated by one or more identities. The identity can be determined based on the TSIG key in the TSIG record.

In the home and also during construction in the professional case, islands of security exist when the authoritative DNS server has no connection to parent or children zones. In a later stage, access can be provided via DNS root servers involving a second security anchor. Alternatively, stub resolvers access contact recursive name servers via an secure channel as SIG(0) [RFC2931] or TSIG [RFC2845].

In spite of using DNSSEC, rogue devices can obtain valid addresses and accept commands for these destinations. This is not worse than rogue devices accepting any packet via a NIC in promiscuous mode. Rogue devices can send unwanted commands to the thus observed IP address. The same addresses can also be obtained by listening to the network traffic.



## 6.2. RD Security considerations

Security for Resource Direcorry is provided by recommended CoAP security techniques: DTLS.

## 7. Acknowledgments

Zach Shelby wrote the original Discovery section in [I-D.ietf-core-coap] which forms the basis for this draft. This I-D has benefited from conversations with and comments from Emmanuel Frimout, Michael Verschoor, Jamie Mc Cormack, Esko Dijk, Dee Denteneer, Joop Talstra, Jerald Martocci, Matthieu Vial, Jerome Hamel, George Yianni, and Nicolas Riou.

Sye Loong Keoh, Sandeep Kumar, and Oscar Garcia Morchon contributed actively to security considerations.

## 8. Changelog

Changes from -02 to -03:

- o adapted to resource direcorry version -03
- o DNS-SD examples follow better DNS-SD naming conventions
- o added gp= link-format attribute
- o added security considerations
- o groups of end-points and groups of Function Sets instead of groups of devices
- o less centered around DNS, more RD aspects

## 9. References

### 9.1. Normative References

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC2845] Vixie, P., Gudmundsson, O., Eastlake, D., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", RFC 2845, May 2000.

- [RFC2931] Eastlake, D., "DNS Request and Transaction Signatures (SIG(0)s)", RFC 2931, September 2000.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.
- [RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, October 2005.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.
- [RFC4605] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")", RFC 4605, August 2006.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011.
- [RFC6206] Levis, P., Clausen, T., Hui, J., Gnawali, O., and J. Ko,

"The Trickle Algorithm", RFC 6206, March 2011.

## 9.2. Informative References

- [I-D.cheshire-dnsext-dns-sd]  
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-11 (work in progress), December 2011.
- [I-D.eggert-core-congestion-control]  
Eggert, L., "Congestion Control for the Constrained Application Protocol (CoAP)", draft-eggert-core-congestion-control-01 (work in progress), January 2011.
- [I-D.ietf-6man-uri-zoneid]  
Carpenter, B. and R. Hinden, "Representing IPv6 Zone Identifiers in Uniform Resource Identifiers", draft-ietf-6man-uri-zoneid-01 (work in progress), May 2012.
- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.ietf-core-groupcomm]  
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-01 (work in progress), March 2012.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-14 (work in progress), June 2012.
- [I-D.lynn-core-discovery-mapping]  
Lynn, K. and Z. Shelby, "CoRE Link-Format to DNS-Based Service Discovery Mapping", draft-lynn-core-discovery-mapping-01 (work in progress), July 2011.
- [I-D.lynn-homenet-site-mdns]  
Lynn, K. and D. Sturek, "Extended Multicast DNS", draft-lynn-homenet-site-mdns-00 (work in progress), March 2012.
- [I-D.shelby-core-coap-req]

Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, "CoAP Requirements and Features", draft-shelby-core-coap-req-02 (work in progress), October 2010.

[I-D.shelby-core-interfaces]

Shelby, Z. and M. Vial, "CoRE Interfaces", draft-shelby-core-interfaces-02 (work in progress), March 2012.

[I-D.shelby-core-resource-directory]

Shelby, Z. and S. Krco, "CoRE Resource Directory", draft-shelby-core-resource-directory-03 (work in progress), May 2012.

[I-D.vanderstok-core-bc]

Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-05 (work in progress), October 2011.

[I-D.jennings-http-srv]

Jennings, C., "DNS SRV Records for HTTP", draft-jennings-http-srv-05 (work in progress), March 2009.

[I-D.giacomin-core-sleepy-option]

Fossati, T., Giacomini, P., Loreto, S., and M. Rossini, "Sleepy Option for CoAP", draft-giacomin-core-sleepy-option-00 (work in progress), February 2012.

[I-D.vial-core-mirror-proxy]

Vial, M., "CoRE Mirror Proxy", draft-vial-core-mirror-proxy-00 (work in progress), March 2012.

[ZeroConf]

Cheshire, S. and D. Steinberg, "Zero Configuration Networking: The Definitive Guide", O'Reilly Media, Inc. , ISBN 0-596-10100-7, 2006.

[UPNP]

"Universal Plug and Play", Web <http://www.upnp.org>, 2012.

Authors' Addresses

Peter van der Stok (editor)  
vanderstok consultancy  
Kamperfoelie 8  
Helmond, 5708 DM  
The Netherlands

Email: [consultancy@vanderstok.org](mailto:consultancy@vanderstok.org)

Kerry Lynn  
Consultant

Phone: +1-978-460-4253  
Email: [kerlyn@ieee.org](mailto:kerlyn@ieee.org)

Anders Brandt  
Sigma Designs  
Emdrupvej 26A, 1.  
Copenhagen O, 2100  
Denmark

Email: [Anders\\_Brandt@sigmadesigns.com](mailto:Anders_Brandt@sigmadesigns.com)



CoRE  
Internet-Draft  
Intended status: Standards Track  
Expires: January 14, 2013

M. Vial  
Schneider-Electric  
July 13, 2012

CoRE Mirror Server  
draft-vial-core-mirror-proxy-01

Abstract

The Constrained RESTful Environments (CoRE) working group aims at realizing the REpresentational State Transfer (REST) architecture in a suitable form for the most constrained nodes. Thanks to the Constrained Application Protocol (CoAP), REST is now applicable to constrained networks. However the most energy-constrained devices may enter sleep mode and disconnect their network link during several minutes to save energy, hence preventing them from acting as traditional web servers. This document describes how a sleeping device can store resource representations in an entity called Mirror Server and participate in a constrained RESTful environment.

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 January 14, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

|   |    |
|---|----|
| 1. Introduction . . . . .                 | 3  |
| 1.1. Motivation . . . . .                 | 3  |
| 1.2. Uses cases . . . . .                 | 3  |
| 1.3. Assumptions and objectives . . . . . | 4  |
| 2. Requirements Language . . . . .        | 5  |
| 3. Architecture . . . . .                 | 5  |
| 4. Mirror Server Function Set . . . . .   | 6  |
| 4.1. Discovery . . . . .                  | 7  |
| 4.2. Registration . . . . .               | 8  |
| 4.3. Update . . . . .                     | 11 |
| 4.4. Validation . . . . .                 | 12 |
| 4.5. Removal . . . . .                    | 12 |
| 4.6. SEP Operation . . . . .              | 12 |
| 4.7. Client Operation . . . . .           | 14 |
| 4.8. Modification check . . . . .         | 15 |
| 5. Acknowledgements . . . . .             | 17 |
| 6. IANA Considerations . . . . .          | 17 |
| 7. Security Considerations . . . . .      | 17 |
| 8. References . . . . .                   | 18 |
| 8.1. Normative References . . . . .       | 18 |
| 8.2. Informative References . . . . .     | 19 |
| Author's Address . . . . .                | 19 |



## 1. Introduction

### 1.1. Motivation

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM, energy harvesting) and networks (e.g. 6LoWPAN). The CoRE charter says that the CoAP protocol [I-D.ietf-core-coap] will support various form of "caching" to support sleeping devices. And the CoAP requirement REQ3 in [I-D.shelby-core-coap-req] clearly states that support of sleeping devices is required:

The ability to deal with sleeping nodes. Devices may be powered off at any point in time but periodically "wake up" for brief periods of time.

As pointed out by [I-D.arkko-core-sleepy-sensors], the server model is not appropriate for the most energy-constrained devices. CoAP also supports the Publish/Subscribe pattern through CoAP observe [I-D.ietf-core-observe]. Notifications with CoAP observe prove to be efficient however as it is currently specified, it still requires the server model to create and maintain the observation relationship. Although CoAP observe may be enhanced to support subscriptions initiated by the observed server, this method is not currently specified. Also in general, a SEP would support only a limited number of observers at a time. The client model is a viable approach but the interactions and interfaces between endpoints are currently undefined. In conclusion, the current working group documents do not propose a complete solution for sleeping devices that are not always reachable.

### 1.2. Uses cases

With the emergence of the Internet of Things we expect a major breakthrough in the number of smart objects in our environment. Yet providing these objects with sufficient energy for continued operation and long battery lifetime is still a big challenge. That is the reason why this specification strives to provide a solution to dramatically reduce the power consumption of constrained RESTful sensors. For battery-operated devices the need to improve battery lifetime is persistent either to reduce the size of smart objects and fit new applications, to increase the product lifetime when it is directly coupled to its battery lifetime or to reduce the annoyance, costs and wastes incurred by changing batteries too frequently. There is also a new trend to avoid batteries and create sensors that can harvest energy from their environment. For those devices it is of prime importance to maintain a high ratio between harvested energy

and power consumption. This ratio has a direct impact on service availability and the user experience especially because the harvesting efficiency is typically not constant in time (e.g day/night for a photovoltaic cell).

### 1.3. Assumptions and objectives

In this specification we assume that the energy-constrained devices can store a sufficient amount of energy to enable bi-directional communication and to perform periodic tasks like maintaining soft state. However the most constrained devices may not be able to store energy and may have unpredictable availability due to sporadic energy production (e.g. self-powered push button). This specification may be applicable to these devices as long as they have enough energy to perform the initial registration. This may require an additional source of power during the commissioning phase.

Throughout this document we will only consider sleeping devices that are totally unreachable during long periods of time. In other word, network connectivity is turned off at least several seconds hence generating unacceptable interruptions if the device runs as a server. Some link-layer technologies offer advanced low power modes such as duty-cycle link activity or receiver initiated transmissions hence allowing the devices to sleep while still offering network connectivity with an always-on illusion. Devices for which the available energy is sufficient to afford always-on illusion are out of scope of this specification since the server model is applicable to these endpoints.

Efficient support of sleeping devices has implications on many aspects of the IP stack: Media Access Control (MAC), neighbor discovery, routing, REST intermediaries... This specification does not aim to find a solution for all of those. The objective is to provide an interaction model at the application level where data exchanges are always initiated by the sleeping endpoint. This way the application can finely control when the network link needs to be on. In no way the mechanisms defined here precludes usage of a low power mode at link-layer.

This specification does not pretend to provide full REST support to sleeping devices. These devices will be provided with the minimum set of REST features to publish resources. Particular attention is paid to facilitate configuration and to associate meta-data to resources from sleeping devices.

## 2. 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 RFC 2119 [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988], [I-D.shelby-core-resource-directory] and [I-D.shelby-core-interfaces]. Readers should also be familiar with the terms and concepts discussed in [I-D.ietf-core-coap] and [I-D.ietf-core-link-format]. This specification makes use of the following additional terminology:

**Sleeping device:** A smart object that can enter a long period of time with its network link in disconnected state in order to save energy.

**Sleeping endpoint (SEP):** A sleeping endpoint is an IP sleeping device which can participate in a constrained RESTful environment as a client-only endpoint.

**Mirror Server (MS):** A server endpoint that implements the Mirror Server Function Set.

## 3. Architecture

The Mirror Server architecture is shown in Figure 1. A Mirror Server (MS) is a web server implementing a special Function Set that allows a sleeping endpoint (SEP) to create resources in the MS resource tree. For energy efficiency a SEP is a client-only CoAP endpoint and hence is not able to serve content by itself. The MS implements REST interfaces allowing a SEP to maintain a set of mirrored resources that will be served in turn by the MS. So a Mirror Server acts as a mailbox between the sleeping endpoint and the client. A CoAP client discovers resources owned by the SEP but hosted on the MS using typical mechanisms such as `"/.well-known/core"` [I-D.ietf-core-link-format] or Resource Directory [I-D.shelby-core-resource-directory].

A SEP must register and maintain a mirror entry on the MS, which is soft state and need to be periodically refreshed. A MS provides interfaces to register, update and remove a mirror entry and an associated set of mirrored resources. Furthermore, a MS provides interfaces to read and update the mirrored resources from both the SEP and client sides. Finally, a mechanism to discover a MS using the CoRE Link Format is defined.

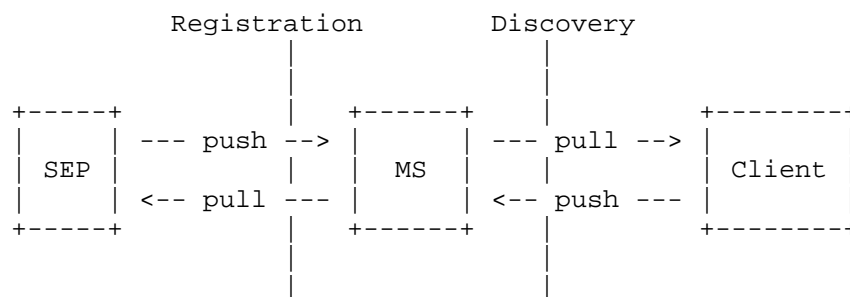


Figure 1: Mirror Server architecture

The Mirror Server functionality can be distributed over multiple server endpoints in the network or centralized on a single server endpoint. A shorter round-trip time gives better energy efficiency for request/response exchanges, so it is important to choose a path between the Mirror Server and the sleeping endpoint with minimum latency. Moreover a sleeping endpoint with a Mirror Server in its direct neighborhood may even avoid having to configure global IP connectivity. However in a wireless network relying on local connectivity may result in fragility due to device mobility or radio fluctuations. This could lead a sleeping endpoint to frequently try to move from one Mirror Server to another. Consequently, clients would need to restart resource discovery frequently. In that regard, a centralized Mirror Server gives more stability. A centralized Mirror Server also concentrates network traffic on a central point and may cause network congestion in a mesh network. However data flow of a sleeping endpoint is expected to be low hence mitigating the risk of network congestion.

A sleeping endpoint MAY register with more than one Mirror Server but in that case the resources of a sleeping endpoint appear duplicated during resource discovery. Section 4.1 describes how to detect duplicate resources.

#### 4. Mirror Server Function Set

The interface is mostly identical to that of the Resource Directory Function Set defined in [I-D.shelby-core-resource-directory] so this specification only points out the differences. Contrary to the Resource Directory there is no lookup Function Set in a Mirror Server. Indeed, from a client point of view, the mirrored resources look like any other resources hosted the MS endpoint. So resource discovery of mirrored resources is directly available through `"/.well-known/core"` instead of a separate Function Set.

The examples presented in this section make use of a smart temperature sensor the resources of which are defined below using Link Format. Three resources are dedicated to the Device Description (manufacturer, model, name) and one contains the current temperature in degree Celsius.

```
</dev/mfg>;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s";obs
```

#### 4.1. Discovery

The interaction between a SEP and a MS is based on the same discovery interface as the Resource Directory except that the Resource Type in the URI template is replaced with "core.ms".

The following example shows a sleeping endpoint discovering a MS using this interface, thus learning that the base MS resource is at /ms.

```

SEP                                                                    MS
|----- GET /.well-known/core?rt=core.ms ----->
|
|<---- 2.05 Content "</ms>; rt="core.ms" -----
|

```

```
Req: GET coap://[ff02::1]/.well-known/core?rt=core.ms
Res: 2.05 Content
</ms>;rt="core.ms"
```

Resource discovery between a client and a MS or a client and a RD needs special care to take into account the fact that resources from a sleeping endpoint might appear duplicated. Clients SHOULD employ 2-step resource discovery by looking up sleeping endpoints AND resource types to detect duplicate resources. Clients MAY use single-step resource discovery only if the SEP can register with no more than one Mirror Server. A client can use the "ep" link attribute as a filter on the "/.well-known/core" resource to retrieve a list of endpoints and detect duplicate sleeping endpoints registered on multiple MSs. A client can use the "ep" type of lookup to do the same on a RD. The result of endpoint discovery is then used to filter out duplicate resources returned from simple resource discovery.

The following example shows a client discovering the sleeping

endpoints and learning that the SEP 0224e8fffe925dcf is registered on two Mirror Servers.

| Client                                  | MS1    | MS2 |
|---|--------|-----|
| ----- GET /.well-known/core?ep=* -----> | -----> |     |
| <----- 2.05 Content "</ms/0>..." -----  |        |     |
| <----- 2.05 Content "</ms/0>..." -----  |        |     |

```

Req: GET coap://[ff02::1]/.well-known/core?ep=*
Res: 2.05 Content
</ms/0>;ep="0224e8fffe925dcf"
Res: 2.05 Content
</ms/0>;ep="02004cffffe4f4f50"
</ms/1>;ep="0224e8fffe925dcf"

```

From the previous exchange and the next resource discovery request, the client can infer that the resources `coap://ms1/ms/0/sen/temp` and `coap://ms2/ms/1/sen/temp` actually come from the same sleeping endpoint.

| Client                                      | MS1    | MS2 |
|---|--------|-----|
| - GET /.well-known/core?rt=ipso:ucum.Cel -> | -----> |     |
| <----- 2.05 Content "</ms/0>..." -----      |        |     |
| <----- 2.05 Content "</ms/1>..." -----      |        |     |

```

Req: GET coap://[ff02::1]/.well-known/core?rt=ucum.Cel
Res: 2.05 Content
</ms/0/sen/temp;rt="ucum.Cel"
Res: 2.05 Content
</ms/1/sen/temp>;rt="ucum.Cel"

```

#### 4.2. Registration

The registration interface is identical to the registration interface of the Resource Directory Function Set except that the preferred path for the Mirror Server Function Set is `/ms`.

After discovering the location of a MS Function Set, a sleeping endpoint MAY register its resources that need to be mirrored using the registration interface. This interface accepts a POST from an

endpoint containing a description of the resources to be created on the Mirror Server as the message payload in the CoRE Link Format along with query string parameters indicating the endpoint identifier, its domain and the lifetime of the registration. The Link Format description is identical to the `"/.well-known/core"` resource found on a typical server endpoint meaning that the Interface Description attributes are actually intended for the Mirror Server. A Mirror Server **MUST** reject a registration if at least one of the Interface Descriptions is not supported. Upon successful registration a MS creates a new resource or updates an existing resource for the mirror entry and returns its location. The resources specified by the SEP during registration are created as sub-resources of the mirror entry on the MS endpoint. The registration interface **MUST** be implemented to be idempotent, so that registering twice with the same endpoint parameter does not create multiple MS entries. The resource associated to a mirror entry **SHOULD** implement the Interface Type CoRE Link List defined in [I-D.shelby-core-interfaces]. A GET request on this resource **MUST** return the list of mirrored resources for the corresponding SEP.

After successful registration, a MS **SHOULD** enable resource discovery for the new mirrored resources by updating its `"/.well-known/core"` resource. A MS **MUST** wait for the initial representation of a mirrored resource before it can be visible in resource discovery. The top level resource corresponding to a mirror entry **MUST** be published in `"/.well-known/core"` to enable 2-step resource discovery described in Section 4.1. Sub-resources of a mirror entry **SHOULD** be discoverable either directly in `"/.well-known/core"` or indirectly through gradual reveal from the mirror entry resource. The Web Link of a mirror entry **MUST** contain an `"ep"` attribute with the value of the End-Point parameter received at registration. If present, the End-Point Type parameter **SHOULD** also be mapped as a `"rt"` attribute.

A Mirror Server **MAY** be configured to register the SEP's resources in a Resource Directory (RD). A SEP **MUST NOT** register the mirrored resources in a RD by itself. It is always the responsibility of the Mirror Server. Since each SEP may register resources with different lifetimes, a MS **MUST** register the resources of a SEP in a separate resource directory entry. A SEP may register with multiple MS hence the RD entries from the different MS for the same SEP would overlap if special care is not taken. Therefore if a SEP is likely to register with more than one MS, a Mirror Server **MUST** create its own domain to register the resources of a SEP. This precaution ensures that the `ep` identifier of a SEP is unique for each domain in the RD. The new domain is typically formed by concatenating the MS's endpoint identifier with the domain in use.

SEP resources in the MS are kept active for the period indicated by

the lifetime parameter. The SEP is responsible for refreshing the entry within this period using either the registration or update interface. Once a mirror entry has expired, the MS deletes all resources associated to that entry and updates its `"/.well-known/core"` resource. When the mirrored resources are also registered in a RD, the RD and MS entries are supposed to have the same lifetime. Consequently, when the mirror entry expires, a MS MAY let the RD entry expire too instead of explicitly deleting it. Nevertheless if the MS entry is deleted using the Removal interface then the RD entry MUST be explicitly removed.

A Mirror Server could lose or delete the mirror entry associated to a SEP without sending an explicit notification (e.g. after reboot). A SEP SHOULD be able to detect this situation by processing the response code while using the SEP Operation or Update interface. Especially an error code "4.04 Not Found" SHOULD cause the SEP to register again. A SEP MAY also register with multiple MSs to alleviate the risk of interruption of service.

Implementation note: It is not recommended to reuse the value of the ep parameter in the URI of the Mirror Server entry. This parameter may be a relatively long identifier to guarantee global uniqueness (e.g. EUI64) and would generate inefficient URIs on the Mirror Server where only a local handler is necessary.

The following example shows a sleeping endpoint registering with a MS.

```
SEP                                                                 MS
```

---

```
|    --- POST /ms "</dev..." ----->|
```

---

```
|                                         |
```

---

```
| <-- 2.01 Created Location: /ms/0 -----|
```

---

Req: POST coap://ms.example.org/ms?ep=0224e8fffe925dcf&rt=sensor

Etag: 0x3f

Payload:

```
</dev/mfg>;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s";obs
```

Res: 2.01 Created

Location: /ms/0

The mirror entry resource below has been created on the MS.



```
Req: GET coap://ms.example.org/.well-known/core
Res: 2.05 Content
</ms>;rt="core.ms",
</ms/0>;ep="0224e8ffffe925dcf";rt="sensor";if="core.ll"
```

The SEP sets the initial value for its mirrored resources and the following resources are now created.

```
Req: GET coap://ms.example.org/ms/0
Res: 2.05 Content
Payload:
</ms/0/dev/mfg>;rt="ipso.dev.mfg";if="core.rp",
</ms/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</ms/0/dev/n>;rt="ipso.dev.n";if="core.p",
</ms/0/sen/temp>;rt="ucum.Cel";if="core.s";obs
```

Then the MS registers the mirrored resources in the RD.

```

MS | RD
|  |
|  | --- POST /rd "</ms/0..." ----->
|  |
|  | <-- 2.01 Created Location: /rd/6534 -----
|  |

```

```
Req: POST coap://rd.example.org/rd?ep=0224e8fffe925dcf&
      rt=sensor&d=msl.example.org
```

```
Etag: 0x6a
Payload:
</ms/0/dev/mfg >|rt="ipso.dev.mfg";if="core.rp",
</ms/0/dev/mdl>|rt="ipso.dev.mdl";if="core.rp",
</ms/0/dev/n>|rt="ipso.dev.n";if="core.p",
</ms/0/sen/temp>|rt="ucum.Cel";if="core.s";obs
```

Res: 2.01 Created  
Location: /rd/6534

### 4.3. Update

The update interface is not necessary on a Mirror Server. A SEP can use the registration interface to modify a mirror entry. The Lifetime query parameter of the SEP operation interface defined in Section 4.6 allows a SEP to extend the lifetime of its mirror entry.

#### 4.4. Validation

The validation interface is not supported on a Mirror Server since the sleeping endpoint is unreachable.

#### 4.5. Removal

The removal interface is identical.

Upon successful removal, `"/.well-known/core"` and the Resource Directory (if applicable) **MUST** be updated accordingly. All resources associated to the mirror entry are deleted.

#### 4.6. SEP Operation

The SEP Operation interface is not defined for a Resource Directory and is specific to the Mirror Server Function Set.

Once the resources have been created on the MS, the SEP can access its mirrored resources at its own pace. The SEP **MAY** update its mirrored resources on the MS using PUT requests. The SEP **MAY** retrieve the current representation of any of its mirrored resources using GET requests. The SEP can reactivate its mirror entry by including a Lifetime (lt) parameter in the query string. While updating dynamic resources, a SEP **SHOULD** include a Lifetime parameter with the smallest value that matches its technical constraints. It allows a client to fastly detect a stale mirror entry. A SEP **MAY** omit processing some responses for non confirmable requests in order to avoid spending energy waiting for a response when it is frequently accessing a mirrored resource. Nevertheless a SEP **SHOULD** periodically check the responses to ensure that its mirror entry is still active on the MS.

Other specifications may override or extend this interface to provide more advanced features, support other REST methods and queuing patterns. This is however out of scope of this specification which provides only a basic behavior.

The basic SEP operation interface is specified as follows:

Interaction: SEP -> MS

Method: GET, PUT

URI Template: `/[+location][+resource][?lt]`

## URI Template Variables:

location := This is the Location path returned by the MS as a result of a successful registration.

resource := This is the relative path to a mirrored resource managed by the registered SEP.

lt := Lifetime (optional). The number of seconds by which the lifetime of the whole mirror entry is extended. Range of 1-4294967295. If no lifetime is included, the current remaining lifetime stays unchanged.

Content-Type: Defined at registration

Etag: The Etag option MAY be included to allow clients to validate a resource on multiple Mirror Servers.

Success: 2.01 "Created", the request MUST include the initial representation of the mirrored resource.

Success: 2.04 "Changed", the request MUST include the new representation of the mirrored resource.

Success: 2.05 "Content", the response MUST include the current representation of the mirrored resource.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example describes how a sleeping endpoint can initialize the resource containing its manufacturer name just after registration.

| SEP |                                     | MS |
|-----|-------------------------------------|----|
|     | --- PUT /ms/0/dev/mfg "acme" -----> |    |
|     |                                     |    |
|     | <-- 2.01 Created -----              |    |

Req: PUT /ms/0/dev/mfg

Payload: acme

Res: 2.01 Created

The example below shows how a SEP can indicate that it is supposed to send a temperature value at least every hour to keep its mirror entry active.

```

SEP                                     MS
|                                     |
| --- PUT /ms/0/sen/temp?lt=3600 "22" -----> |
|                                     |
| <-- 2.04 Changed ----- |
|                                     |

Req: PUT /ms/0/sen/temp?lt=3600
Payload: 22
Res: 2.04 Changed

```

#### 4.7. Client Operation

The Client Operation interface is not defined for a Resource Directory and is specific to the Mirror Server Function Set.

While the SEP operation interface describes only the interaction between the SEP and the MS on mirrored resources, the interface between a client and the MS for mirrored resources is actually defined by the Link Format payload at registration. This specification does not define the list of Interface Description attribute values that a Mirror Server must support. This is left to a companion specification such as a CoRE profile specification. A Mirror Server MAY support complex interfaces that require special logic and interactions between multiple mirrored resources. The CoRE Batch interface defined in [I-D.shelby-core-interfaces] is an example of complex interface that defines relations between a parent resource and sub-resources using SenML [I-D.jennings-senml].

A SEP may register resources with the "obs" attribute. In that case a MS using the CoAP protocol [I-D.ietf-core-coap] SHOULD accept to establish a CoAP observation relationship between the observable mirrored resource and a client as defined in [I-D.ietf-core-observe].

A SEP may stop updating its mirrored resources without explicitly removing its mirror entry (e.g. transition to another MS after network unreachability detection). A client can detect this situation when the corresponding mirror entry has expired. Upon receipt of a response with error code 4.04 "Not Found", a client SHOULD restart resource discovery to determine if the resources are now mirrored on another MS.

The Client operation interface is specified as follows:

Interaction: Client -> MS

Method: Defined at registration

URI Template: `/[+location][+resource]`

URI Template Variables:

`location` := This is the Location path returned by the MS as a result of a successful registration.

`resource` := This is the relative path to a mirrored resource managed by a SEP.

Content-Type: Defined at registration

In the example below a client observes the changes of temperature through the Mirror Server.

| SEP                          | MS                                   | Client |
|------------------------------|--------------------------------------|--------|
|                              | <- GET /ms/0/sen/temp -<br>(observe) |        |
|                              | -- 2.05 Content "22" ->              |        |
| - PUT /ms/0/sen/temp "23" -> |                                      |        |
| <- 2.04 Changed -----        |                                      |        |
|                              | -- 2.05 Content "23" ->              |        |

#### 4.8. Modification check

This interface is not defined for a Resource Directory and is specific to the Mirror Server Function Set.

A sleeping endpoint may register resources supporting POST or PUT methods and therefore that could be modified by clients. In that case the SEP needs to poll these resources to detect updates. Polling each modifiable resource is inefficient when they are numerous. The modification check interface allows a SEP to retrieve a list of resources that have been modified. The SEP can then send GET requests on each resource of the list to get the updated representation. A POST request on the check interface automatically clears the list of modified resources.

The check interface is specified as follows:

Interaction: SEP -> MS

Method: POST

URI Template: /{+location}?chk

URI Template Variables:

location := This is the Location path returned by the MS as a result of a successful registration.

Request Content-Type: None

Response Content-Type: application/link-format

Success: 2.04 "Changed", the response MUST include a list of web links to resources that have been modified by clients.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows a commissioning tool changing the name of a sleeping device through a Mirror Server. A commissioning button on the SEP triggers the reading of the new configuration.

| SEP                      | MS                      | Client |
|--------------------------|-------------------------|--------|
|                          | <-- PUT /ms/0/dev/n --- |        |
|                          | -- 2.04 Changed ----->  |        |
| [press button on SEP]    |                         |        |
| - POST /ms/0?chk ----->  |                         |        |
| <- 2.04 Changed -----    |                         |        |
| - GET /ms/0/dev/n -----> |                         |        |
| <- 2.05 Content -----    |                         |        |

Req: PUT /ms/0/dev/n  
 Payload: "sensor-1"  
 Res: 2.04 Changed

Req: POST /ms/0?chk  
 Res: 2.04 Changed  
 Payload: "</ms/0/dev/n>"

Req: GET /ms/0/dev/n  
 Res: 2.05 Content  
 Payload: "sensor-1"

## 5. Acknowledgements

Thanks to Zach Shelby who is the author of the Resource Directory interface. Thanks to Nicolas Riou, Jari Arkko, Esko Dijk who have provided fruitful comments.

## 6. IANA Considerations

"core.ms" resource type needs to be registered.

The "ep" attribute needs to be registered.

## 7. Security Considerations

This document needs the same security considerations as described in Section 7 of [RFC5988] and Section 6 of [I-D.ietf-core-link-format].

The Mirror Server architecture defines the SEP and Client roles in the Mirror Function Set interfaces. Since the roles are based on the requester identity, a MS SHOULD perform appropriate authentication in order to prevent a malicious client endpoint from impersonating the SEP or an authorized client. Otherwise the malicious client could gain access to interfaces allowing corruption or deletion of a mirrored resource.

A malicious client could start a denial of service attack by trying to mirror a large resource on a MS. Memory exhaustion would prevent other sleeping endpoints from mirroring their resources. A MS SHOULD use quotas to limit the size and the number of mirrored resources per SEP.

A Mirror Server is actually an intermediary running at application level. As a consequence the Mirror Server architecture can only provide implicit end-to-end security that relies on a trusted network if security is not available at application layer. When explicit end-to-end security is required between a SEP and a Client, data object security SHOULD be employed.

## 8. References

### 8.1. Normative References

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-10 (work in progress), June 2012.
- [I-D.ietf-core-link-format]  
Shelby, Z., "CoRE Link Format",  
draft-ietf-core-link-format-14 (work in progress),  
June 2012.
- [I-D.ietf-core-observe]  
Hartke, K., "Observing Resources in CoAP",  
draft-ietf-core-observe-05 (work in progress), March 2012.
- [I-D.shelby-core-interfaces]  
Shelby, Z. and M. Vial, "CoRE Interfaces",  
draft-shelby-core-interfaces-03 (work in progress),  
July 2012.
- [I-D.shelby-core-resource-directory]  
Shelby, Z. and S. Krco, "CoRE Resource Directory",  
draft-shelby-core-resource-directory-03 (work in progress),  
July 2012.



progress), May 2012.

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

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

## 8.2. Informative References

[I-D.arkko-core-sleepy-sensors]  
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.

[I-D.jennings-senml]  
Jennings, C., Shelby, Z., and J. Arkko, "Media Types for Sensor Markup Language (SENML)", draft-jennings-senml-08 (work in progress), January 2012.

[I-D.shelby-core-coap-req]  
Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, "CoAP Requirements and Features", draft-shelby-core-coap-req-02 (work in progress), October 2010.

## Author's Address

Matthieu Vial  
Schneider-Electric  
Grenoble,  
FRANCE

Phone: +33 (0)47657 6522  
Email: matthieu.vial@schneider-electric.com



CoRE  
Internet-Draft  
Intended status: Informational  
Expires: April 15, 2013

L. Wang  
W. Wang  
BUPT  
L. Zhu  
F. Yu  
Huawei Technologies  
October 12, 2012

CoAP Option Extensions: Profile and Sec-flag  
draft-wang-core-profile-secflag-options-02

Abstract

This memo adds two Options for the Constrained Application Protocol (CoAP): Profile and Sec-flag. The Profile Option is indicating the identification of an application using CoAP. The Sec-flag Option complements the security considerations of CoAP.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 15, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

|  |    |
|--|----|
| 1. Introduction . . . . .                                  | 3  |
| 1.1. Terminology . . . . .                                 | 3  |
| 2. Motivations . . . . .                                   | 3  |
| 2.1. Profile Option Extension . . . . .                    | 3  |
| 2.2. Sec-flag Option Extension . . . . .                   | 4  |
| 3. Profile Option . . . . .                                | 4  |
| 3.1. Profile Option Definition . . . . .                   | 5  |
| 3.1.1. Option Value Length . . . . .                       | 5  |
| 3.2. Using the Profile Option . . . . .                    | 6  |
| 3.3. Example . . . . .                                     | 6  |
| 4. Sec-flag Option . . . . .                               | 7  |
| 4.1. Security Negotiation . . . . .                        | 7  |
| 4.2. Using the Sec-flag Option in data transport . . . . . | 9  |
| 4.3. Sec-flag Option Definition . . . . .                  | 10 |
| 4.4. System Overview . . . . .                             | 10 |
| 5. Security Considerations . . . . .                       | 11 |
| 6. IANA Considerations . . . . .                           | 11 |
| 7. References . . . . .                                    | 11 |
| 7.1. Normative Reference . . . . .                         | 11 |
| 7.2. Informative References . . . . .                      | 11 |
| Authors' Addresses . . . . .                               | 12 |

## 1. Introduction

CoAP is a specialized web transfer protocol for machine-to-machine applications such as smart energy and building automation using with constrained nodes and networks. This memo adds two new options for CoAP: Profile and Sec-flag.

The main purpose of the Profile Option is indicating the identification of an application using CoAP, by reading this option some intermediaries (e.g. proxy) and transport networks could distinguish different applications and do some differentiated processing.

The Sec-flag Option complements the security considerations, enabling NoSec pattern in a segment of the communication path between the client and server, by taking care of establishing and maintaining lower layer security instead of DTLS in these intermediate networks.

### 1.1. Terminology

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

## 2. Motivations

CoAP is a light-weight web protocol and can be used in constrained devices, fulfilling machine-to-machine requirements. Because of its features, more and more M2M applications MAY adopt CoAP.

### 2.1. Profile Option Extension

CoAP applications SHOULD use an operator's network as the transport bearer. Different machine-to-machine applications MAY have different Quality of Service (QoS) requirements in terms of required bit rates as well as acceptable packet delays and packet loss rates. When application data is transmitted through the transport network, the network MAY need to identify different machine-to-machine services to do some differentiated processing, applying different control policies with subscriptions. Before applying control policies to applications, transport networks SHOULD identify them and distinguish each one from another referring to application identification, and then networks MAY apply different policies to different applications. Some intermediaries (e.g. CoAP proxy) MAY also would like to distinguish different applications and do some differentiated processing such as caching and forwarding application data in different priorities.

This memo describes the extensions to CoAP and is to provide expanding proposal(s) to fulfill the motivations and requirements, defining an additional Option for the Constrained Application Protocol (CoAP): Profile. The Profile Option is defined as the identification of CoAP applications. When CoAP messages are transmitted through the transport network, network entities MAY use some technologies to read the Option Value to identify the application, and then apply control policies with the subscription of application owner.

## 2.2. Sec-flag Option Extension

The transmission path between the client and server MAY consist of some segments: Network domain based on existing standards 3GPP, TISPAN, IETF, etc., and M2M Device and Gateway Domain based on existing standards and technologies like DLMS, CEN, CENELEC, PLT, Zigbee, M-BUS, KNX, etc. The application data MAY be transmitted through different networks between the client and server.

The basic CoAP protocol defines the DTLS binding. DTLS would add a per-datagram overhead and some initialization vectors. For some constrained networks and nodes, this is really expensive. And intermediate network domain MAY have some independent and reliable security standards (e.g. ZigBee standard). In some cases, CoAP could use these security standards instead of DTLS to avoid DTLS overhead in some intermediate networks. In these network domains DTLS may be disabled but be retained in other domains.

As an example, the ZigBee standard for sensor networks defines a security architecture based on an online trust center and uses CCM\* model to secure applications. This standard can fulfill the security requirements of CoAP. That is to say, CoAP applications could be secured by lower layer security, so in this network DTLS could be disabled to avoid DTLS datagram overhead. We just mark a security flag to indicate that CoAP data is secured by lower layer instead in this network domain and the overhead would be much less. And in the Transport Network domain we still establish DTLS security. Thus we MAY enable two different security patterns described in [I-D.ietf-core-coap] in different segments between the client and server.

The Sec-flag Option can be used to indicate the security information and ensure the integrity of the security mechanism.

## 3. Profile Option

### 3.1. Profile Option Definition

| No. | C/E      | Name    | Format      | Length | Default |
|-----|----------|---------|-------------|--------|---------|
| 2n  | Elective | Profile | (see below) | 4B     | (none)  |

The Profile Option indicates the identification of CoAP applications. Transport network entities MAY use some technologies to read the Option Value and then apply corresponding treatment.

This option is "elective" and the Option Number is even. It MUST NOT occur more than once.

The detailed definitions and encoding SHOULD refer to the description of Option Format in [I-D.ietf-core-coap]. It is RECOMMENDED that the Option Value consists of Enterprise Number, Application ID and Priority.

| Enterprise Number | Application ID | Pri |
|-------------------|----------------|-----|
|-------------------|----------------|-----|

Figure 1: Option Value Format

As shown in Figure 1, Enterprise Number is the register number of application owners (e.g. traffic management agencies) in network operators. Application ID is the identification of the owner's application which subscribes transport and communication services from operators. Priority (Pri) indicates the priority of application data, data of the same application MAY has different priority (For some cost reasons, application owners MAY subscribe low priority for some application data).

The SDNV[RFC5050] encoding can be used.

We can distinguish different applications with a combination of Enterprise Number, Application ID and Priority.

#### 3.1.1. Option Value Length

In actual usages, the number of CoAP application owners MAY be out of length range indicated by 2 bytes, the default length cannot fulfill requirements. Hence, we can define another Option Value Length: 5bytes.

In the initialization phase of CoAP message, the Option Value Length SHOULD be determined.

### 3.2. Using the Profile Option

The semantics of Option Value are defined by prior agreement between the application owners and network operators. Some encryption algorithms MAY be used. Network entities MAY also apply some validation policies when reading the Option Value.

CoAP application owners MAY realize functions through a M2M communication for some purposes (e.g. Meter readings) at their customer's premises. The Profile Option can be contained in the application message to indicate the identity.

When CoAP messages across transport network, network entities MAY use some technologies such as Deep Packet Inspection (DPI) to read the Profile Option Value and report it to policy control decision function entities. And then policy control decision function entities determine the policies applied to CoAP data as well as establishing dedicated bearers.

### 3.3. Example

In some M2M environments, the nodes access to Internet through 3GPP network.

This example (Figure 2) shows that mobile network is the bearer between two M2M end-points. These end-points MAY belong to an electric power company and this M2M application is a meter reading service. The profile option value of this application is 0x12111111.

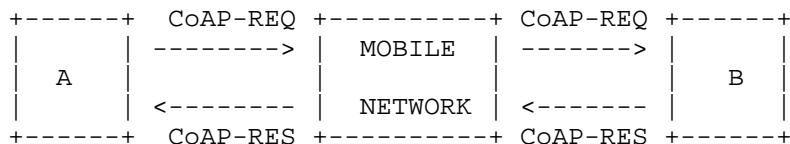


Figure 2 An example

The message flow is shown in Figure 3. At first the requester (server) sends a request which MAY be a device trigger that makes end devices return electricity records. The number of end devices is numerous and this triggering MAY happen in a preset period of time. All devices return their records at the approximately same time, and the data transmission volume is huge. Hence it is expected that the network SHOULD offer QoS guarantee (such as high bandwidth and



throughput) for the M2M application.

The requester SHOULD initial the Profile Option when sending a request. Network entities can read the Option Value and know that the application is a meter reading service belonging to a certain electricity company. And then specialized policies MUST be applied. The Profile option value MUST be echoed in the messages from recipients.

When the M2M application data comes into the network, network entities MUST provide corresponding policy control with subscriptions and MAY also establish dedicated bearers assign dedicated network resources to ensure the quality of transport and communication if necessary.

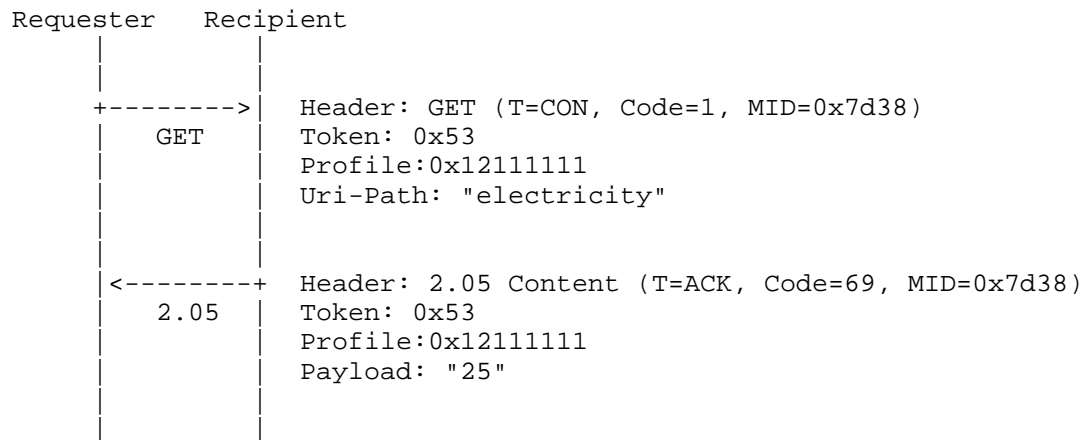


Figure 3 Profile Option in CoAP messages

#### 4. Sec-flag Option

The Sec-flag Option complements the security considerations, enabling NoSec pattern in one or more segments of the communication path between the client and server.

##### 4.1. Security Negotiation

Before establishing a security session between endpoints, the negotiation SHOULD be made. As shown in Figure 4, the basic model includes three actors: a client, a proxy and a server.

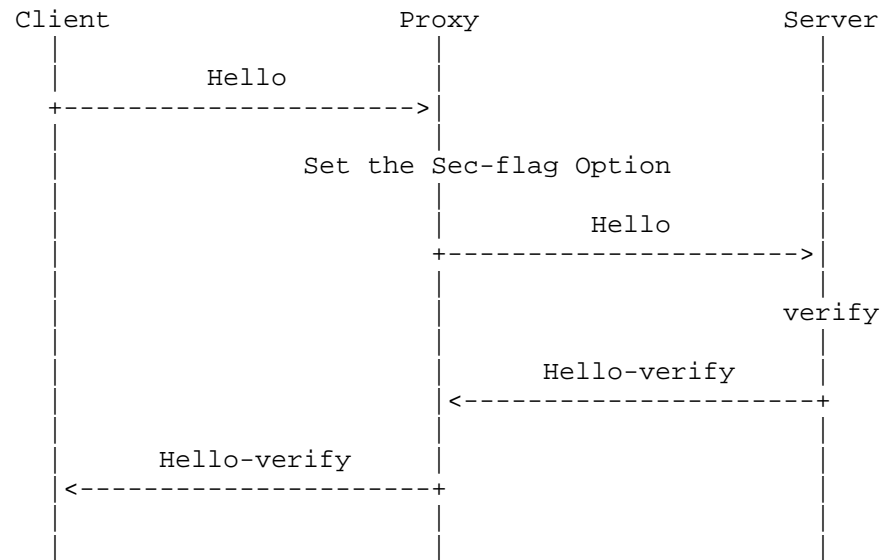


Figure 4 Basic model

(1) Lower security can secure CoAP application data

At first, the client sends to the server a hello message in which the Sec-flag Option with an empty value is included. The proxy is requested to forward the request or serve it, and return the response. If the network domain between the client and proxy could guarantee the lower layer security, the proxy SHOULD set the Sec-flag Option with a valid value and transfer the hello message to the server.

When the server receives the message, it MUST respond with a server hello-verify message. A response with the same valid option value as the value set in the proxy SHOULD be returned only if the server trusts the lower layer security between the client and proxy. If the server accepts the lower layer security, DTLS would be disabled between the client and proxy and then the proxy SHOULD make a DTLS handshake with the server to make up a DTLS security session. Otherwise, the DTLS handshake SHOULD be made between the client and server, that would be introduced in the following.

(2) Lower security cannot secure CoAP application data

When the server receives the Hello message, if the server does not trust the lower layer security between the client and proxy, the server would respond a hello-verify message containing an empty

Security option and an error code. Then the client would send DTLS handshake messages to the server and establish DTLS between the client and the server.

(3) There is no lower security between the client and M2M Gateway

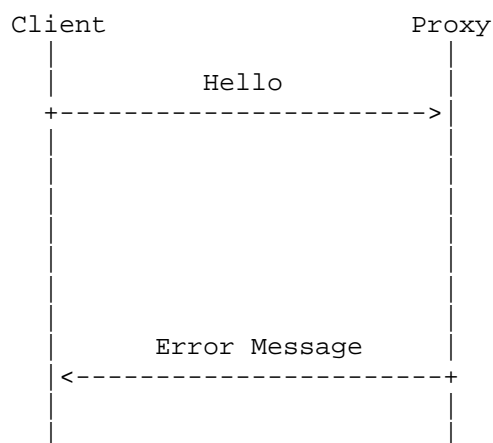


Figure 5 There is no lower security in the Device and Gateway Domain

If the network domain between the client and proxy could not guarantee lower layer security, the proxy SHOULD return an appropriate error response code with an empty Security Option as shown in Figure 5. Then the client would send DTLS handshake messages to the server and establish DTLS between the client and the server.

In all the situations, the client and the proxy also need initialize lower security mechanism, which MAY happen either before the Security Option negotiation or after it.

#### 4.2. Using the Sec-flag Option in data transport

When the security negotiation is completed, a security session is established between the client and server.

The Sec-flag Option MUST be included in messages sent by the client and the value SHOULD be empty. When the proxy receives the message, it MUST set the Sec-flag Option with a valid value and transfer the message to the server. The Sec-flag Option indicates that the message comes from a reliable network domain and the server could trust it. And then the server would respond the request. The same Sec-flag Option SHOULD be echoed in the response. When the response

comes to the proxy, the proxy reads the option and knows that the message SHOULD be secured by lower layer security.

#### 4.3. Sec-flag Option Definition

| No.  | C/E      | Name     | Format      | Length | Default |
|------|----------|----------|-------------|--------|---------|
| 2n+1 | Critical | Sec-flag | (see below) | 1B     | (empty) |

The Sec-flag Option is used for indicating the lower layer security.

This option is "critical" and the Option Number is odd.

The detailed definitions and encoding SHOULD refer to the description of Option Format in [I-D.ietf-core-coap]. The value is made up of security indication.

The SDNV[RFC5050] encoding can be used.

#### 4.4. System Overview

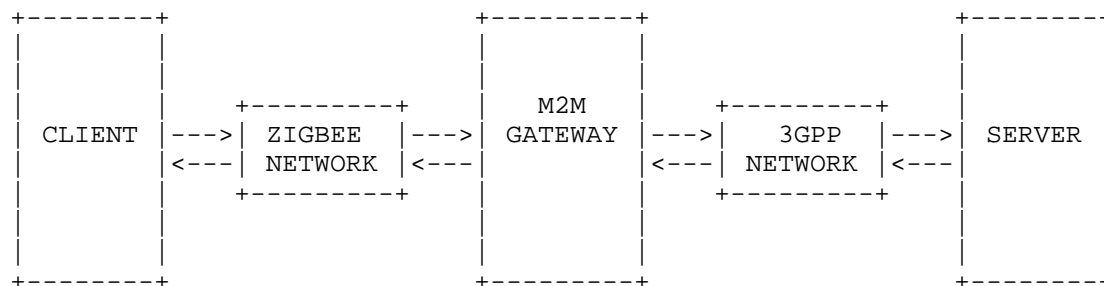


Figure 6 System overview of a usage scenario

As shown in Figure 6, the nodes access to Internet through 3GPP network. DTLS is enabled between M2M Gateway and application server, and in Zigbee network DTLS is disabled and lower layer security secures CoAP applications. M2M Gateway, working as a "marker", sets the Sec-flag option to show that the data from Zigbee network domain is secured and reliable in the uplink and indicate that the data need low layer security and DTLS SHOULD be disabled in the Zigbee network domain in the downlink. As the intermediate gateway, M2M Gateway needs to transfer the data to each network and adapt the security standard to the other one. M2M Gateway also needs to check the

message. When a message from the end devices is received, the gateway checks whether the Sec-flag option is set by the end devices illegally (the default value SHOULD be empty). If the Sec-flag option is not empty, the gateway SHOULD drop it, else the gateway SHOULD set the valid value and transfer the message.

## 5. Security Considerations

To be defined.

## 6. IANA Considerations

The following entries are added to the CoAP Option Numbers registry:

| Number | Name     | Reference |
|--------|----------|-----------|
| 2n     | Profile  | RFC XXXX  |
| 2n+1   | Sec-flag | RFC XXXX  |

## 7. References

### 7.1. Normative Reference

- [I-D.ietf-core-coap]  
Shelby, Z., Hartke, K., Bormann, C., and B. Frank,  
"Constrained Application Protocol (CoAP)",  
draft-ietf-core-coap-12 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol  
Specification", RFC 5050, November 2007.

### 7.2. Informative References

- [I-D.fossati-core-publish-monitor-options]  
Fossati, T., Giacomini, P., and S. Loreto, "Publish and  
Monitor Options for CoAP",  
draft-fossati-core-publish-monitor-options-01 (work in  
progress), March 2012.

## Authors' Addresses

Lei Wang  
Beijing University of Posts and Telecommunications  
Xitucheng road 10  
Haidian District, Beijing 100876  
P. R. China

Email: [wleibblue@163.com](mailto:wleibblue@163.com)

Wendong Wang  
Beijing University of Posts and Telecommunications  
Xitucheng road 10  
Haidian District, Beijing 100876  
P. R. China

Email: [wdwang@bupt.edu.cn](mailto:wdwang@bupt.edu.cn)

Lei Zhu  
Huawei Technologies  
Huawei Building, Q20 No.156 Beiqing Rd.Z-park  
Haidian District, Beijing 100095  
P. R. China

Email: [lei.zhu@huawei.com](mailto:lei.zhu@huawei.com)

Fang Yu  
Huawei Technologies  
Huawei Building, Q20 No.156 Beiqing Rd.Z-park  
Haidian District, Beijing 100095  
P. R. China

Email: [grace.yufang@huawei.com](mailto:grace.yufang@huawei.com)

