Network Working Group J. Arkko
Internet-Draft A. Keranen
Intended status: Informational Ericsson
Expires: January 27, 2012 July 26, 2011

CoAP Security Architecture
draft-arkko-core-security-arch-00

Abstract

Constrained Application Protocol (CoAP) is a light-weight protocol designed to be used in machine-to-machine applications. This memo describes challenges associated with securing CoAP and proposes a new security model that the authors believe is suitable for these environments. The model requires minimal amount of configuration, but still provides strong security and is a natural fit with the typical communication practices smart object networking environments. This memo also proposes JSON payload format extensions to support the architecture.

Table of Contents

1. Introduction

Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] is a light-weight protocol designed to be used in machine-to-machine applications such as smart energy and building automation.

This memo describes implementation and operational challenges associated with securing CoAP in these environments (Section 3), reviews related work in solving these challenges (Section 2), and proposes a security model (Section 4) that the authors believe is suitable for many machine-to-machine application environments. The model requires minimal amount of configuration, but still provides strong security and is a natural fit with the typical communication practices smart object networking environments. Finally, this memo proposes some protocol and payload format extensions to support the architecture (Section 5). Section 6 provides a summary of the approach.

2. Related Work

CoAP base specification [I-D.ietf-core-coap] outlines how to use DTLS [RFC5238] and IPsec [RFC4306] for securing the protocol. DTLS can be applied with group keys, pairwise shared keys, or with certificates. The security model in all cases is mutual authentication, so while there is some commonality to HTTP in verifying the server identity, in practice the models are quite different. The specification says little about how DTLS keys are managed.

The IPsec mode is described with regards to the protocol requirements, noting that small implementations of IKEv2 exist [I-D.kivinen-ipsecme-ikev2-minimal]. However, the specification is silent on policy and other aspects that are normally necessary in order to implement interoperable use of IPsec in any environment [RFC5406].

[I-D.garcia-core-security] discusses the overall security problem for Internet of Things devices. It also discusses various solutions, including IKEv2/IPsec [RFC4306], TLS/SSL [RFC5246], DTLS [RFC5238], HIP [RFC5201] [I-D.ietf-hip-rfc5201-bis] [I-D.moskowitz-hip-rg-dex], PANA [RFC5191], and EAP [RFC3748]. The draft also discusses various operational scenarios, bootstrapping mechanisms, and challenges associated with implementing secity mechanisms in these environments.

[I-D.iab-smart-object-workshop] gives an overview of the security discussions at the March 2011 IAB workshop on smart objects. The workshop recommended that additional work is needed in developing suitable credential management mechanisms (perhaps something similar

to the Bluetooth pairing mechanism), understanding the implementability of standard security mechanisms in small devices (see, for instance, [I-D.kivinen-ipsecme-ikev2-minimal]), and additional research in the area of lightweight cryptographic primitives.

[I-D.sarikaya-core-sbootstrapping] discusses the bootstrapping problem with low-powered nodes, and argues that this problem should be solved at a general level and not left to link layer specific mechanisms. The draft looks at EAP [RFC3748], PANA [RFC5191], HIP Diet Exchange (HIP-DEX) [I-D.moskowitz-hip-rg-dex], and 802.1X [IEEE.802-1X.2010] as potential solutions for bootstrapping.

[I-D.moskowitz-hip-rg-dex] defines a light-weight version of the HIP protocol for low-power nodes. This version uses a fixed set of algorithms, elliptic curve cryptography, and eliminates hash functions. The protocol still operates based on host identities, and runs end-to-end between hosts, protecting IP layer communications. [RFC6078] describes an extension of HIP that can be used to send upper layer protocol messages without running the usual HIP base exchange at all.

[I-D.daniel-6lowpan-security-analysis] makes a comprehensive analysis of security issues related to 6LOWPAN networks, but its findings also apply more generally for all low-powered networks. Some of the issues this document discusses include the need to minimize the number of transmitted bits and simplify implementations, threats in the smart object networking environments, and the suitability of 6LOWPAN security mechanisms, IPsec, and key management protocols for implementation in these environments.

Cryptographically Generated Addresses (CGAs) [RFC3972] and Host Identity Protocol (HIP) [RFC5201] have employed similar ideas as those proposed in this memo, though with slightly different purpose in mind, and at a different protocol layer. Similarly, PGP [RFC4880] and other similar tools have popularized the concept of exchanging key fingerprint values off-line. This is very similar to what is proposed in this memo.

[I-D.rescorla-jsms], [I-D.jones-json-web-signature], and [I-D.jones-json-web-token] propose JSON extensions similar to those discussed in this memo, though constructed for other purposes. Further work is needed to analyze if these proposals could be used as a basis for smart object security communication security as well. Obviously, general-purpose JSON signature mechanisms should be used if they exist, even if some additional data elements might have to be defined to carry all the information that this memo requires.

3. Challenges

This section discusses three challenges: implementation difficulties, practical provisioning problems, and layering and communication models.

The most often discussed issues in the security for the Internet of Things relates to implementation difficulties. The desire to build small, battery-operated, and inexpensive devices drives the creation of devices with a limited protocol and application suite. Some of the typical limitations include running CoAP instead of HTTP, limited support for security mechanisms, limited processing power for long key lengths, sleep schedule that does not allow communication at all times, and so on. In addition, the devices typically have very limited support for configuration, making it hard to set up secrets and trust anchors.

The implementation difficulties are important, but they should not be overemphasized. It is important to select the right security mechanisms and avoid duplicated or unnecessary functionality. But at the end of the day, if strong cryptographic security is needed, the implementations have to support that. Also, the use of the most lightweight algorithms and cryptographic primitives is useful, but should not be the only consideration in the design. Interoperability is also important, and often other parts of the system, such as key management protocols or certificate formats are heavier to implement than the algorithms themselves.

The second challenge relates to practical provisioning problems. These are perhaps the most fundamental and difficult issue, and unfortunately often neglected in the design. There are several problems in the provisioning and management of smart object networks:

- Small devices have no natural user interface for configuration that would be required for the installation of shared secrets and other security-related parameters. Typically, there is no keyboard, no display, and there may not even be buttons to press. Some devices may only have one interface, the interface to the network.

- Manual configuration is rarely, if at all, possible, as the necessary skills are missing in typical installation environments (such as in family homes).

- There may be a large number of devices. Configuration tasks that may be acceptable when performed for one device may become unacceptable with dozens or hundreds of devices.

- Network configurations evolve over the lifetime of the devices, as additional devices are introduced or addresses change. Various central nodes may also receive more frequent updates than individual devices such as sensors embedded in building materials.

Finally, layering and communication models present difficulties for straightforward use of the most obvious security mechanisms. Smart object networks typically pass information through multiple participating nodes [I-D.arkko-core-sleepy-sensors] and end-to-end security for IP or transport layers may not fit such communication models very well. The primary reasons for needing middleboxes relates to the need to accommodate for sleeping nodes as well to enable the implementation of nodes that store or aggregate information.

4. Proposed Architecture

The proposed security architecture describes both a deployment model for provisioning as well as a technical model for networks and protocols.

The basis of the architecture are self-generated secure identities, similar to Cryptographically Generated Addresses (CGAs) [RFC3972] or Host Identity Tags (HITs) [RFC5201]. That is, we assume the following holds:

$$I = h(P|O)$$

where I is the secure identity of the device, h is a hash function, P is the public key from a key pair generated by the device, and O is optional other information.

4.1. Provisioning

As provisioning security credentials, shared secrets, and policy information is difficult, the provisioning model is based only on the secure identities. A typical network installation involves physical placement of a number of devices while noting the identities of these devices. This list of short identifiers can then be fed to a central server as a list of authorized devices. Secure communications can then commence with the devices, at least as far as information from from the devices to the server is concerned, which is what is needed for sensor networks. Actuator networks and server-to-device communication is covered in Section 4.4.

Where necessary, the information collected at installation time may also include other parameters relevant to the application, such as

the location or purpose of the devices. This would enable the server to know, for instance, that a particular device is the temperature sensor for the kitchen.

Collecting the identity information at installation time can be arranged in a number of ways. The authors have employed a simple but not completely secure method where the last few digits of the identity are printed on a tiny device just a few millimeters across. Alternatively, the packaging for the device may include the full identity (typically 32 hex digits), retrieved from the device at manufacturing time. This identity can be read, for instance, by a bar code reader carried by the installation personnel. (Note that the identities are not secret, the security of the system is not dependent on the identity information leaking to others. The real owner of an identity can always prove its ownership with the private key which never leaves the device.) Finally, the device may use its wired network interface or proximity-based communications, such as Near-Field Communications (NFC) or Radio-Frequency Identity tags (RFIDs). Such interfaces allow secure communication of the device identity to an information gathering device at installation time.

No matter what the method of information collection is, this provisioning model minimizes the effort required to set up the security. Each devices generates its own identity in a random, secure key generation process. The identities are self-securing in the sense that if you know the identity of the peer you want to communicate with, messages from the peer can be signed by the peer's private key and it is trivial to verify that the message came from the expected peer. There is no need to configure an identity and certificate of that identity separately. There is no need to configure a group secret or a shared secret. There is no need to configure a trust anchor. In addition, the identities are typically collected anyway for application purposes (such as identifying which sensor is in which room). Under most circumstances there is actually no additional configuration effort from provisioning security.

4.2. Device Groups

In some deployment cases it is also possible to configure the identity of an entire group of devices, rather than registering the individual devices. For instance, many installations employ a kit of devices bought from the same manufacturer in one package. It is easy to provide an identity for such a set of devices as follows:

Idev = h(Pdev|Potherdev1|Potherdev2|...|Potherdevn)

Igrp = h(Pdev1|Pdev2|...|Pdevm)

where Idev is the identity of an individual device, Pdev is the public key of that device, and Potherdevi are the public keys of other devices in the group. Now, we can define the secure identity of the group (Igrp) as a hash of all the public keys of the devices in the group (Pdevi).

The installation personnel can scan the identity of the group from the box that the kit came in, and this identity can be stored in a server that is expected to receive information from the nodes. Later when the individual devices contact this server, they will be able to show that they are part of the group, as they can reveal their own public key and the public keys of the other devices. Devices that do not belong to the kit can not claim to be in the group, because the group identity would change if any new keys were added to Igrp.

4.3. Protocol Architecture

As noted above, the starting point of the architecture is that nodes self-generate secure identities which are then communicated out-of-band to the peers that need to know what devices to trust. To support this model in a protocol architecture, we also need to use these secure identities to implement secure messaging between the peers, explain how the system can respond to different types of attacks such as replay attempts, and decide at what protocol layer and endpoints the architecture should use.

Securing the messages is straightforward. A node with identity I should sign each message it sends with the private key associated with the identity I. This allows the recipient to verify that the message was constructed by the sender. This is similar to what Secure Neighbor Discovery (SEND) does with its RSA Signature Option [RFC3971].

However, this simple model needs some enhancements to be able to withstand denial-of-service and replay attacks. As we expect connectivity in smart object networks to be intermittent, traditional active methods such as nonce exchanges are not suitable. Instead, an optional timestamp-based approach SHOULD be used in addition to the basic signatures. This approach is similar to the one used to secure unsolicited SEND messages. Nodes that implement the timestamp approach need to have a real-time clock or they need to synchronize to one using a network time protocol [RFC5905]. Additionally, nodes that have persistent memory, SHOULD implement a monotonically increasing sequence number. Message recipients SHOULD silently ignore messages when they see a timestamp value that is out of range from the current time plus or minus a small time drift factor. Similarly, recipients that have seen multiple messages from the same sender SHOULD silently ignore messages that do not have a sequence

number greater than the one they have seen last.

These exchanges are basic cryptographic protocol tools, and have been used in different layers of the IP protocol stack for different purposes. For instance, HIP in its opportunistic mode could be used to implement largely the same functionality at the IP layer. However, it is our belief that the right layer for this solution is at the application layer. More specifically, in the data formats transported in the payload part of CoAP. This approach provides the following benefits:

- Ability for intermediaries to act as caches to support different sleep schedules, without the security model being impacted.

- Ability for intermediaries to be built to perform aggregation, filtering, storage and other actions, again without impacting the security of the data being transmitted or stored.

- Ability to operate in the presence of traditional middleboxes, such as a protocol translators or even NATs (not that we recommend their use in these environments).

Note that there is no requirement that the secure identities be associated with IP addresses. They can certainly be used as input material for constructing addresses for stateless address autoconfiguration [RFC4862], but this is not required.

4.4. Actuator Networking

The above architecture is a perfect fit for sensor networks where information flows from large number of devices to small number of servers. But it is not sufficient alone for other types of applications. For instance, in actuator applications a large number of devices need to take commands from somewhere else. In such applications it is necessary to secure that the commands come from an authorized source.

This can be supported, with some additional provisioning effort and optional pairing protocols. The basic provisioning approach is as described in Section 4.1, but in addition there must be something that informs the devices of the identity of the trusted server(s). There are multiple ways to provide this information. One simple approach is to feed the identities of the trusted server(s) to devices at installation time. This requires either a separate user interface, local connection (such as USB), or using the network interface of the device for configuration. In any case, as with sensor networks the amount of configuration information is minimized: just one short identity value needs to be fed in. Not both an

identity and a certificate. Not shared secrets that must be kept confidential. An even simpler provisioning approach is that the devices in the device group discussed in Section 4.2 trust each other. Then no configuration is needed at installation time.

When both peers know the expected cryptographic identity of the other peer off-line, secure communications can commence.

Alternatively, various pairing schemes can be employed. Note that these schemes can benefit from the already secure identifiers on the device side. For instance, the server can send a pairing message to each device after their initial power-on and before they have been paired with anyone, encrypted with the public key of the device. As with all pairing schemes that do not employ a shared secret or the secure identity of both parties, there are some remaining vulnerabilities that may or may not be acceptable for the application in question.

In any case, the secure identities help again in ensuring that the operations are as simple as possible. Only identities need to be communicated to the devices, not certificates, not shared secrets or IPsec policy rules.

5. Proposed Protocol Extensions

The concrete implementation of the proposed architecture involves a specification for the identity format and generation, and a specification of the data format necessary to carry the signature, public key, timestamp, and sequence number data objects.

The data format part of this specification could be implemented in various ways, as S/MIME data [RFC3851], XML signatures [RFC3275], or as additional data in JSON [I-D.jennings-senml] [RFC4627]. We have chosen to use the JSON format in this memo.

5.1. Identity Format

The format of identifiers in binary representation is 128-bit identifiers. These identifiers have no association with any existing number space managed by IANA. In particular, they are not part of the IPv6 address space; they exist at application layer.

The identifiers can be represented in textual form as Universal Resource Names (URNs), with the format "device:cgi-HEX" where "device" is the designated new URN type, "cgi" is a subtype that stands for cryptographically generated identifiers, and HEX is an exactly 32 characters long string of hex digits.

While not at the right layer from the point of view of our architecture, these identities could also be used in the Authority Name part of CoAP DTLS (Section 10 of [I-D.ietf-core-coap]), IKE or other lower-level protocols.

5.2. Identity Generation

The process of generating a new identity takes two input values: the public key of the identity owner as a DER-encoded ASN.1 structure of the type SubjectPublicKeyInfo, and optional other parameters.

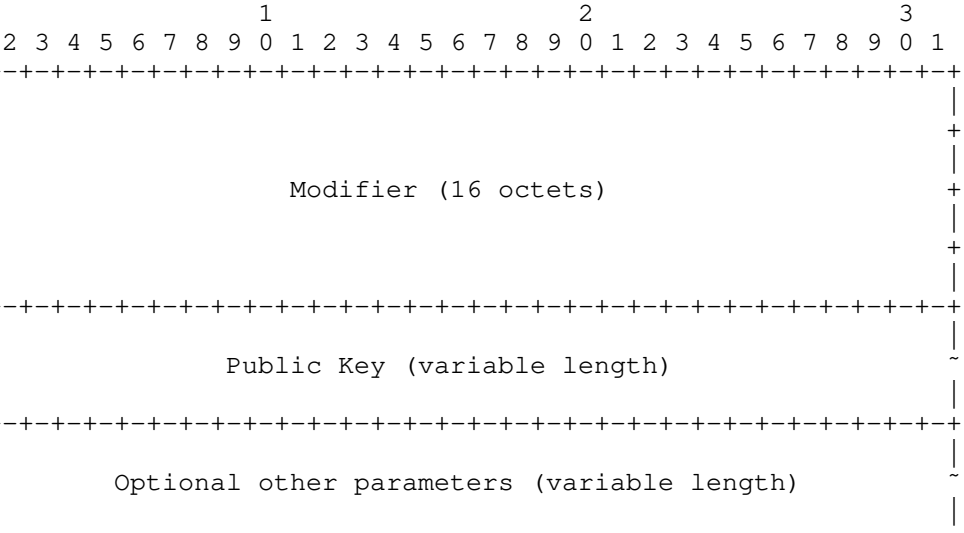
An identity and associated Identity Parameters Block (defined further below) SHOULD be generated as follows:

1. Generate a modifier, a random or pseudo-random 128-bit value.

2. Concatenate from left to right the modifier value, the encoded public key, and any optional other parameters. Execute the SHA-256 algorithm [FIPS.180-3.2008] on the concatenation. Take the 128 leftmost bits of the SHA-256 hash value. The result is the identity.

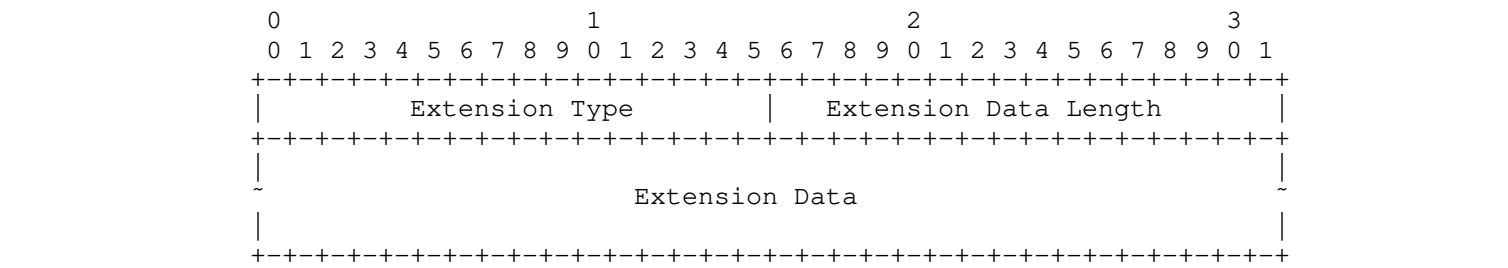
3. Form an Identity Parameters Block data structure by concatenating from left to right the modifier value, the encoded public key, and any optional other parameters.

The output of the address generation algorithm is a new identity and a new Identity Parameters Block data structure. The latter data structure has the following format:



The Public Key field MUST be formatted as a DER-encoded [CCITT.X690.2002] ASN.1 structure of the type SubjectPublicKeyInfo, defined in the Internet X.509 certificate profile [RFC3280]. RSA public/private key pair SHOULD be used. When RSA is used, the algorithm identifier MUST be rsaEncryption, which is 1.2.840.113549.1.1.1, and the RSA public key MUST be formatted by using the RSAPublicKey type as specified in Section 2.3.1 of RFC 3279 [RFC3279].

The other parameters is a sequence of extension blocks with the following format:



Where

Extension Type

16-bit identifier of the type of the Extension Field. Identifier for the one currently defined extension is defined in Section 5.2.1, and some reserved values and values for testing use are given in Section 8. The summary of the defined values is as follows:

Value	Name
0x0000	Reserved (Section 8)
0x0001	Identifier_Group (Section 5.2.1)
0xFFFD	Exp_FFFD (Section 8)
0xFFFE	Exp_FFFE (Section 8)
0xFFFF	Exp_FFFF (Section 8)

Extension Data Length

16-bit unsigned integer. Length of the Extension Data field of this option, in octets.

Extension Data

Variable-length field. Extension-Type-specific data.

5.2.1. Identifier Groups

This extension has the Extension Type 0x0001 (Identifier_Group). The purpose of the extension is to carry the public keys of other devices in a group of devices. As discussed in Section 4.2, this can be used to show membership of a group and ease the provisioning process.

The extension data should consist of a 16-bit length field that expresses the number of public keys that follow, followed by each public key, encoded as described in Section 5.2.

5.3. JSON Identity

Messages that employ secure identities and carry JSON [RFC4627] payloads need to carry information about the identity of the device that ultimately provided the payload. This information is necessary to understand the source of the information, and is also necessary to verify a cryptographic signature attached to the payload. However, the mechanisms for transporting information about the identity and making a signature are kept separate.

An identity is represented by a two-field object in JSON, for instance:

```
{ "id": "device:cgi-27611bc81020716627ff0000cfaa1234",
  "ipb":  "4e26b808cd05d4e26b80912ae3e26b809143fe4e26b4GFTR35f8266" }
```

The "id" field MUST be included, and an additional "ipb" field for the Identity Parameters Block MAY be included. To save communications bandwidth, the optional field MAY be omitted even when the sender has the information. However, the "ipb" field SHOULD appear frequently enough in messages that recipients have likely cached it.

5.3.1. The id Field

This field MUST contain an identity string in the format defined in Section 5.1.

5.3.2. The ipb Field

This field MUST contain the BASE64-encoded Identity Parameters Block associated with the same identity as given in the "id" field.

5.4. JSON Signature Envelope

Messages that employ secure identities and carry JSON [RFC4627] payloads need to carry enough information to prove that the message came from the right source. The JSON Signature Envelope is a JSON object that carries a signature. Together with the JSON identity fields it becomes possible for the recipients to verify the signature. This object can be used to implement secure communication for devices that have the secure identifiers described above and that use JSON to transport information. Other signature envelope formats are needed for other payload formats, but the authors believe that the JSON format is widely applicable to smart objects.

Note that multiple competing ways to represent signature envelopes in JSON are under development [I-D.rescorla-jsms], [I-D.jones-json-web-signature], and [I-D.jones-json-web-token]. The exact choice of encoding remains to be determined; this memo provides its own signature envelope format only for completeness.

Every secure message MUST carry a JSON envelope object. This object MUST have exactly one "jmsg" field for the actual payload, "jid" field for the identity, and "jsig" field for the signature. The fields MUST also appear in this order. The messages MAY carry an additional "jts" field for the timestamp, and "jsq" field for the sequence number. If these fields are included, they MUST appear after the mandatory fields and in the given order.

For instance, the following example contains a JSON signature envelope and a JSON payload from a temperature sensor:

```
 { "jmsg": { "temp": 27.5 },
   "jid":  { "id": "device:cgi-27611bc81020716627ff0000cfaa1234",
             "ipb":  "4e26b808cd05d4e26b912ae3e26b809143fe4eb4GFTR35f82" },
   "jts":  { "s": 1311176727, "f": 123987 },
   "jsq":  23,
   "jsig": "18929abqxc67juil7ff231000912927755bRRwlkadbfddceab"}
```

Note that signatures envelopes can be nested; a JSON signature envelope can be placed inside another signature envelope in the "jmsg" field and signed. This is useful to implement secure intermediaries that want to include additional information beyond what the device itself provided.

5.4.1. The jmsg Field

This field MUST contain the actual payload that the device wants to send, in the usual JSON format.

Note that the JSON envelope needs to be useful without securing information in the rest of the CoAP message carrying it, as well as in situations where it is retransmitted in CoAP or HTTP via an intermediary. For this reason all the relevant information MUST be in the payload part. This is usually the case when taking an information centric approach as in [I-D.arkko-core-sleepy-sensors]. The jid field carries the identity of the device, and the jmsg carries all relevant information about what the devices wants to communicate. Consequently, the payload SHOULD be self-contained, without reference to the source or destination IP addresses of the CoAP message, or to the CoAP/HTTP method or URI.

5.4.2. The jid Field

This field MUST contain an identity as defined in Section 5.3.

5.4.3. The jts Field

This field MUST contain an object with two fields. The first field, "s", indicates the number of seconds since January 1, 1970, 00:00 UTC. At least 48 bits of accuracy is required. The second field, "f" indicate the number of 1/64K fractions of a second, with 16 bits of accuracy.

Implementation note: This format is compatible with the usual representation of time under UNIX, although the number of bits available for the integer and fraction parts may vary.

5.4.4. The jsq Field

This field MUST contain an integer representing a monotonically increasing sequence number of all messages sent by the sender. At least 32 bits of accuracy are required.

5.4.5. The jsig Field

This field MUST contain a variable-length string containing a BASE64-encoded PKCS#1 v1.5 signature, constructed by using the sender's private key over the following sequence of octets:

1. The 128-bit CGI Usage Discriminator value for this specification, 0x53eb e540 4a92 5517 57b6 e398 7aaf a085. (The value has been generated randomly by the editor of this specification.)

2. The entire JSON payload, verbatim and in text as carried in the message, with the contents of the jsig field set to an empty string (jsig: "").

The signature value is computed with the RSASSA-PKCS1-v1_5 algorithm and SHA-256 hash, as defined in [PKCS.1.1993]. Senders use their private key associated with the claimed identity. The "jsig" field MUST be the last one in JSON payload. The resulting PKCS#1 v1.5 signature is put in the "jsig" field.

Receivers MUST treat messages without the "jsig" field as unsecured. A received "jsig" field MUST be checked as follows:

- The receiver MUST ignore any fields that come after the first "jsig" field, for both verification and other processing purposes.
- There must be an associated JSON identity information, so that both the identity and associated public key must be apparent from the secured message, or learned from a preceding message.
- The "jsig" field MUST have correct encoding.
- The signature verification MUST show that the signature has been calculated as specified above.

Messages that do not pass all the above tests MUST be silently discarded if the host has been configured to accept only secured CoAP messages. The messages MAY be accepted if the host has been configured to accept both secured and unsecured messages but MUST be treated as an unsecured message. The receiver MAY also otherwise silently discard packets (e.g., as a response to an apparent CPU exhausting DoS attack).

6. Concluding Remarks

This memo has presented a deployment model, security architecture, and an initial sketch of protocol design to support the architecture. To recap, the main benefits of this model are

- Minimal configuration: per device or per group registration of identities in a server, but no configuration in every device.
- Support for deployment models that are easily implementable by installation personnel. The necessary practices are already employed in typical current smart object networks, even when there is particular support for security.
- Architecture that naturally supports information-centric networking, multicast, middleboxes, aggregation, sleeping nodes, and other aspects that are typical for networking for smart objects.

7. Security Considerations

This entire memo deals with security issues. Some analysis of the security of the mechanisms proposed in this memo is necessary, however.

The security of the architecture rests on the choice of the number of bits in the identifier and the used hash and signature algorithm. With the use of 128 bits identifiers and SHA-256 and RSA, it is expected that the security level is similar to the one in HIP, and goes beyond the 59 bit security of CGAs.

The basic architecture concerns itself only with integrity and data origin verification, not about confidentiality. Where confidentiality or identity privacy is required, additional mechanisms are needed.

Replay attacks can be prevented beyond a small time window of acceptable clock drift, when devices employ the optional timestamp mechanism. This rests on the assumption of secure time synchronization or configuration in the nodes, however. Where NTP is used, its security properties in different modes are discussed in Section 15 of [RFC5905]. In general, no major security problems have been experienced with NTP protocol or reference implementation [NTP.Wikipedia], but protection against determined hostile attackers does require authentication at NTP the layer. Alternative, simpler approaches include relying on the accuracy of clocks set at manufacturing time.

The optional sequence number mechanism can prevent all replay attacks for persistent communications between two peers. Without the use of these two mechanisms there is no support for preventing replay attacks. This may be acceptable in some environments, but not in all.

Any information centric communication model is resistant to attacks against nodes only sending information, as they are not expected to process any security-related messages. Thus, the "sleep torture deprivation attack" described by Stajano and Anderson in [Resurrecting-Duckling] and other denial-of-service attacks of the same nature are not applicable in the architecture proposed in this memo. However, by the same token nodes that receive information become more vulnerable to denial-of-service attacks, as nonce exchanges, puzzles and other standard protocol mechanisms are not used to guard against the receiver having to verify a cryptographic operation on a received packet. The authors believe that this is the right tradeoff for sensor networking, given that server and gateway implementations are more likely to have the necessary capabilities to

deal with attacks than sensor nodes.

8. IANA Considerations

IANA should reserve the new URN type "device" (Section 5.1). A new registry should be created to hold subtypes of this URN type, with the initial value "cgi" defined in this memo. New values can be created through IETF Review or IESG Approval [RFC5226].

IANA should also create a new registry for Cryptographically Generated Identifiers, and add a new name space Extension Type (Section 5.2) there. Policy for adding new extensions in this registry is RFC Required or IESG Approval [RFC5226]. Initial values for the Extension Type field are given below. Assignments consist of a name and the value.

Extension Type 0x0000 should be marked as reserved. Section 5.2.1 allocates Extension Type 0x0001. As recommended in [RFC3692], this document also makes the following assignments for experimental and testing use: the value 0xFFFD, with name Exp_FFFD; the value 0xFFFE, with name Exp_FFFE, and the value 0xFFFF, with name Exp_FFFF.

IANA should also add another new name space to the same registry, for 128-bit CGI Usage Discriminators. These values are allocated on a First Come, First Served basis [RFC5226]. The one initial value in the registry is given in Section 5.4.5.

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-06 (work in progress), May 2011.

[I-D.jennings-senml]
Jennings, C., "Media Type for Sensor Markup Language (SENML)", draft-jennings-senml-05 (work in progress), March 2011.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[PKCS.1.1993] RSA Laboratories, "RSA Encryption Standard, Version 1.5", PKCS 1, November 1993.

[FIPS.180-3.2008] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-3, October 2008, <http://csrc.nist.gov/publications/fips/fips180-3/fips180-3.pdf>.

[CCITT.X690.2002] International International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.

9.2. Informative References

[RFC3275] Eastlake, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, March 2002.

[RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, January 2004.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC3851] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 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.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, November 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.

[RFC5201] Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, "Host Identity Protocol", RFC 5201, 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.

[RFC5406] Bellovin, S., "Guidelines for Specifying the Use of IPsec Version 2", BCP 146, RFC 5406, February 2009.

[RFC6078] Camarillo, G. and J. Melen, "Host Identity Protocol (HIP) Immediate Carriage and Conveyance of Upper-Layer Protocol Signaling (HICCUPS)", RFC 6078, 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.daniel-6lowpan-security-analysis] Park, S., Kim, K., Haddad, W., Chakrabarti, S., and J. Laganier, "IPv6 over Low Power WPAN Security Analysis", draft-daniel-6lowpan-security-analysis-05 (work in progress), March 2011.

[I-D.garcia-core-security]
Garcia-Morchon, O., Keoh, S., Kumar, S., Hummen, R., and R. Struik, "Security Considerations in the IP-based Internet of Things", draft-garcia-core-security-02 (work in progress), July 2011.

[I-D.iab-smart-object-workshop]
Tschofenig, H. and J. Arkko, "Report from the 'Interconnecting Smart Objects with the Internet' Workshop, 25th March 2011, Prague", draft-iab-smart-object-workshop-01 (work in progress), July 2011.

[I-D.ietf-hip-rfc5201-bis]
Moskowitz, R., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", draft-ietf-hip-rfc5201-bis-06 (work in progress), July 2011.

[I-D.jones-json-web-signature]
Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "JSON Web Signature (JWS)", draft-jones-json-web-signature-02 (work in progress), April 2011.

[I-D.jones-json-web-token]
Jones, M., Balfanz, D., Bradley, J., Goland, Y., Panzer, J., Sakimura, N., and P. Tarjan, "JSON Web Token (JWT)", draft-jones-json-web-token-05 (work in progress), July 2011.

[I-D.kivinen-ipsecme-ikev2-minimal]
Kivinen, T., "Minimal IKEv2", draft-kivinen-ipsecme-ikev2-minimal-00 (work in progress), February 2011.

[I-D.moskowitz-hip-rg-dex]
Moskowitz, R., "HIP Diet EXchange (DEX)", draft-moskowitz-hip-rg-dex-05 (work in progress), March 2011.

[I-D.rescorla-jsms]
Rescorla, E. and J. Hildebrand, "JavaScript Message Security Format", draft-rescorla-jsms-00 (work in progress), March 2011.

[I-D.sarikaya-core-sbootstrapping]
Sarikaya, B., Ohba, Y., Moskowitz, R., Cao, Z., and R.

Cragie, "Security Bootstrapping of Resource-Constrained Devices", draft-sarikaya-core-sbootstrapping-02 (work in progress), June 2011.

[IEEE.802-1X.2010]
Institute of Electrical and Electronics Engineers, "IEEE 802.1X Port-Based Network Access Control", IEEE IEEE Standard 802.1X, February 2010.

[Resurrecting-Duckling]
Stajano, F. and R. Anderson, "The Resurrecting Duckling: Security Issues for Ubiquitous Computing", IEEE Computer Journal Volume 42, Issue 5, 2002.

[NTP.Wikipedia]
Wikipedia, "Network Time Protocol", Wikipedia article , July 2011, <http://en.wikipedia.org/wiki/Network_Time_Protocol>.

Appendix A. Acknowledgments

The authors would like to thank to Oscar Novo, Heidi-Maria Rissanen, Samita Chakrabarti, and Fredrik Garneij for interesting discussions in this problem space.

Authors' Addresses

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2012

Z. Cao
China Mobile

July 4, 2011

Allways-online Requirement for Sleeping CoAP Node
draft-cao-core-aol-req-00

Abstract

CoAP is to enable a concept of web of sensors, but there are many cases that the sensors are not always online and hence the requests from the web client cannot reach them in timely manner. This document analyzes this problem and describes the requirements for a CoAP enabled sensor to behave always online.

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 5, 2012.

Table of Contents

1. Introduction

CoAP [I-D.ietf-core-coap] is an Application Protocol for Constrained Nodes/Networks. It is intended to provide RESTful services like 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.

In the Web environment built from HTTP, the resources are distributed among the web servers and clients get/post resources from these always-online servers. Different from HTTP, the usual use case for CoAP is deploy the CoAP server on the tiny sensors and have the Web client visit the resources on the sensor.

On normal cases where the web client can identify the CoAP sensor and the sensor is also alive, CoAP can work perfect. But there are cases that the sensor is unreachable from the ourside network, e.g., as in Figure 1. In this situation, the client cannot directly post/get information from the sensor, and the only way to walk around is to have the sensor proactively connects to the web first of all.

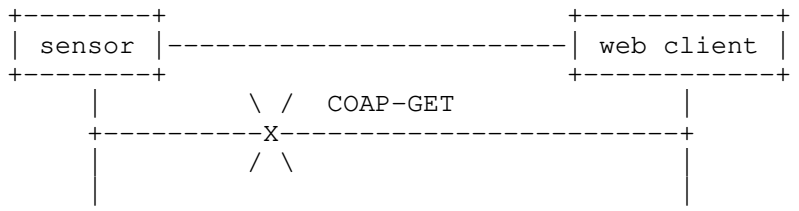


Figure 1: Idle CoAP Sensor Server

In the constrained network environment, this may happen occasionally because the sensors are too constrained to be always online, including the following cases:

1. Sleepy nodes: the tiny sensors that are battery supplied may occasionally fall asleep caused by duty circled MAC or higher layer energy constrained mechanisms.

2. Celluar access nodes: if the sensor directly connects to the cellular network with a celluar modular, it may occasionally lose its IP address dure the timeout mechanism on GPRS connections. The network side can hardly wake up the sensor node from the IP network. The only way is to have the sensor node wake up and connect to the network proactively.

3. NAT constraint: if the sensor is behind an NAT device (either IPv4 NAT or IPv6 NAT), and the mapping relationship between the inter and outer address retires, the Internet client cannot reach the sensor either. The fact that CoAP messages are carried over UDP makes the situation worse, because the UDP mapping timeouts more frequently on the NAT boxes.

In all the above cases, if the web client wants to get the updated information from the sensors, there should be mechanisms to provide the network layer connectivity to enable the CoAP functions. Or in another word, the sensor should behave always online in order to build a real web-of-sensor environment.

This documents analyzes this problem and describes the requirements for a CoAP enabled sensor to behave always online.

1.1. Conventions 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 [RFC2119]

2. Requirements

We described the requirements for CoAP always online as follows:

1. Support of the GET method. When the web client wants to retrieve the most updated information on the sensor, the GET request can reach the sensor within a reasonable delay. Note that for frequently changed states, the coap observe [I-D.ietf-core-observe] can be used to register the subject changes on-demand. But still, the initial CoAP observe message needs to reach the sensor whenever requested.
2. Support of the POST method. When the web client requests the representation be processed by the CoAP server, it sends the POST method to the corresponding URI. In order to behave always online, the POST method request should be able to reach the sensor within a reasonable delay.
3. Support of the PUT method. The PUT method requests that the resource identified by the request URI be updated or created with the enclosed representation. It requires that the PUT method request sent by the web client be processed by sensor server properly.
4. Support of the DELETE method. The DELETE method requests that the resource identified by the request URI be deleted. It requires that the PUT method request sent by the web client be processed by sensor server properly.

5. Minimize complexity. In line with the constrained environment, it requires that the method to make the sensor always online minimize the complexity. For example, it should avoid repeated polling or keep-alive messages.

3. Posible Solutions

4. Security Considerations

TBD.

5. IANA Considerations

This document does not require any IANA actions.

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-06 (work in progress), May 2011.

[I-D.ietf-core-observe]
Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-02 (work in progress), March 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.

Author's Address

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: Informational
Expires: September 15, 2011

A. Castellani
University of Padova
S. Loreto
Ericsson
March 14, 2011

Best Practice to map HTTP to COAP and viceversa
draft-castellani-core-http-coap-mapping-01.txt

Abstract

This draft aims at being a simple guide to the use of CoAP REST interface, to show how it can be mapped to and from HTTP, and at being a base reference documentation for CoAP/HTTP proxy implementors.

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 15, 2011.

Table of Contents

1. Introduction

Since implementing on constrained devices the full HyperText Transfer Protocol (HTTP) [RFC2616] is believed to be operationally and computationally too complex, especially in an M2M communication environment, resources available on constrained nodes are expected to be served using CoAP [I-D.ietf-core-coap].

> "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 (called an end-point). 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." Section 2 [I-D.ietf-core-coap]

These days the information is increasingly converging on the Web, thus an easy CoAP interoperability with HTTP is a paramount feature for CoAP. Indeed leveraging on both the easy CoAP/HTTP translation and the common usage of URI(s) to identify resources, it will become extremely simple to integrate constrained nodes in the Web.

The internetworking described in this document between CoAP and HTTP is mainly based on three points:

- the URI does not change between CoAP and HTTP, the scheme identifies the protocol;
- HTTP/CoAP mapping is performed by a proxy, both HTTP/CoAP endpoints can be not aware that a mapping is happening;
- using a named URI authority and DNS can be useful for the mapping.

The proxy itself does not require any particular knowledge about the constrained network topology, devices contained, nor about the content of data exchanged.

2. HTTP-CoAP

HTTP-CoAP mapping spans across several protocol layers:
- HTTP is mapped to CoAP
- TCP is used on the HTTP side, while CoAP uses UDP transport

In addition to this 6LoWPAN adaptation layer addresses a similar networking scenario, thus a convertion between IPv4/IPv6 to 6LoWPAN MAY be present as well.

2.1. URI

Any resource available in CoAP can be accessed using HTTP at the same URI, except for the scheme. The scheme represents the protocol used by the endpoint to access the resource.

The CoAP resource "//node.coap.something.net/foo" can be accessed using CoAP at the URI "coap://node.coap.something.net/foo", and using HTTP at the URI "http://node.coap.something.net/foo". When the resource is accessed using HTTP, the mapping from HTTP to CoAP is performed by a proxy

The usage of the same URI to access a resource, independently if it is accessed by a CoAP client within the same constrained network or by a HTTP client outside the constrained network, reduces the complexity of a proxy performing the mapping.

OPEN ISSUE: discuss the DNS usage resolving the URI.

2.2. Proxy

A device providing cross-protocol HTTP-CoAP mapping is called HTTP-CoAP cross-protocol proxy (HC proxy).

Usually regular HTTP proxies are same-protocol proxies, because can map from HTTP to HTTP. CoAP same-protocol proxies are intermediaries for CoAP to CoAP exchanges, however the discussion about that entities is out-of-scope of this document.

At least two different kinds of HC proxies may exist:

- One-way cross-protocol proxy (1-way proxy): It can translate from a client of a protocol to a server of another protocol but not viceversa.
- Two-way (or bidirectional) cross-protocol proxy (2-way proxy): It can translate from a client of both protocols to a server of the other protocol.

1-way and 2-way HC proxies can be realized using the following general types of proxies:

Forward proxy (F): It is a proxy known by the client (either CoAP or HTTP) used to access a specific cross-protocol server (respectively HTTP or CoAP). Main feature: server(s) do not require to be known in advance by the proxy (ZSC: zero server configuration).

Reverse proxy (R): It is known by the client to be the server, however for a subset of resources it works as a proxy, by knowing the real server(s) serving each resource. When a cross-protocol resource is accessed by a client, the request will be silently forwarded by the reverse proxy to the real server (running a different protocol). If a response is received by the reverse proxy, it will be mapped, if possible, to the original protocol and sent back to the client. Main feature: client(s) do not require to be known in advance by the proxy (ZCC: zero client configuration).

Transparent (or Intercepting) proxy (I): This proxy can intercept any origin protocol request (HTTP or CoAP) and map it the destination protocol, without any kind of knowledge about the client or server involved in the exchange. Main feature: client(s) and server(s) do not require to be known in advance by the proxy (ZCC and ZSC).

The proxy can be placed in the network at three different logical locations:

Server-side proxy (SS): a proxy placed on the same network domain of the server;

Client-side proxy (CS): a proxy placed on the same network domain of the client;

External proxy (E): a proxy placed in a network domain external to both endpoints.

In the most common scenario the HC proxy is expected to be server-side and deployed at the edge of the constrained network. The arguments supporting this assumption are the following:

TCP/UDP: Translation between HTTP and CoAP requires also a TCP to UDP mapping; UDP performance over the Internet may not be adequate, UDP should be dropped as soon as possible to minimize the number of required retransmissions and overall reliability.

Multicast: To enable access to local-multicast in the constrained network, the HC proxy may require a network interface directly attached to the constrained network.

Caching: Efficient caching requires that all the CoAP traffic is intercepted by the same proxy, network edge is a strategical placement for this need.

Security: HTTPS sessions should be terminated as near as possible to the CoAP server.

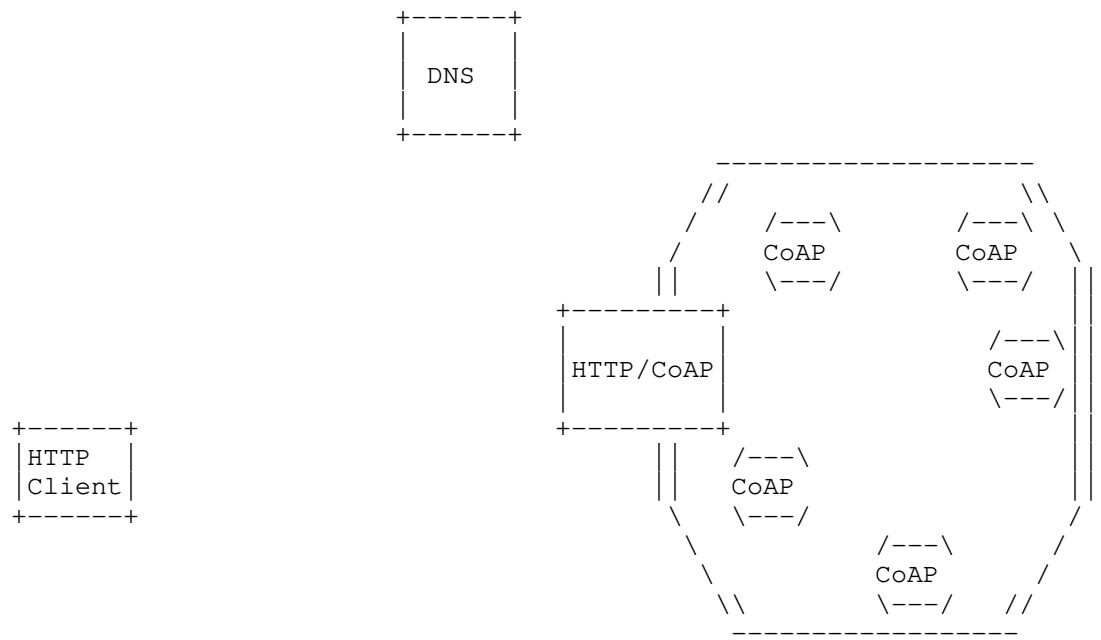


Table 1 shows some interesting HC proxy scenarios, and quickly marks the advantages related to each scenario.

Feature	F CS	R SS	I SS
TCP/UDP	-	+	+
Multicast	-	+	+
Caching	-	+	+
Security	?	+	-
Scalability	+	?	+
Configuration	-	-	+

Table 1: Interesting HC proxy deployments

The following open questions are left open in Table 1:
1. Are CoAP security modes adequate for Internet-wide operation?
2. Are reverse proxy setups scalable?

2.2.1. HC proxy discovery using DNS-SD

DNS-SD can be used by an HTTP client to discover the HC proxy in authority for a specific domain [I-D.jennings-http-srv].

An HTTP client wants access a resource that it knows being identified by the following URI:

//node.coap.something.net/foo

To find the address of the HC proxy, the HTTP client will look up the following SRV record:

_http._tcp.node.coap.something.net

The DNS will contain the following record:

```
      _http._tcp.node.coap.something.net  IN  SRV      0  1  80  hc-proxy.somet
hing.net
        hc-proxy.something.net  IN  A  192.168.0.1  ;  the address of the HC pr
oxy
```

The client will pass the request to the HC proxy that will translate it in a CoAP request. The CoAP side of the proxy will lookup the DNS in order to find the actual constrained device in authority for that URI.

2.3. Mapping

CoAP offers a subset of HTTP features in terms of methods, statuses and options supported; thus some HTTP request MAY NOT be mappable to CoAP.

In particular CoAP lacks the following methods defined in HTTP: OPTIONS, HEAD, TRACE and CONNECT.

An HC proxy receiving an HTTP request with a method not supported in CoAP MUST immediately drop handling the request and MUST send a response with status "405 Method Not Allowed" to the HTTP client.

The mapping of a CoAP response code to HTTP is not straightforward, this mapping MUST be operated accordingly to Table 4 of [I-D.ietf-core-coap].

The mapping of conditional HTTP requests is defined in Section 8.2 of [I-D.ietf-core-coap].

An HC proxy MUST always try to resolve the URI authority, and SHOULD prefer using the IPv6 resolution if available. The authority section of the URI is thus used internally by the HC proxy and SHOULD not be mapped to CoAP.

If an empty CoAP ACK is received, the actual CoAP response is

deferred. As described in CoAP specification the ACK is transparent to the HTTP client.

No upper bound is defined for a server to provide that response, thus for long delays the HTTP client or any other proxy in between MAY timeout, further considerations are available in Section 7.1.4 of [I-D.ietf-httpbis-p1-messaging].

If the HTTP client times out and drops the HTTP session to the proxy (closing the TCP connection), the HC proxy SHOULD wait for the response and cache it if possible. Further idempotent requests to the same resource can use the result present in cache, or if a response has still to come requests will wait on the open CoAP session.

Safe or non-idempotent requests MAY timeout. How the HC proxy should handle this situation?

The HC proxy MUST define an internal timeout for each CoAP request pending, because the CoAP server MAY silently die before completing the request. This timeout SHOULD be as high as possible.

Figure 2 shows an HTTP client on IPv4 (C) accessing a CoAP server on IPv6 (S) through an HC proxy on IPv4/IPv6 (P).
node.coap.something.net has an A record containing the IPv4 address of the HC proxy, and an AAAA record containing the IPv6 of the CoAP server.

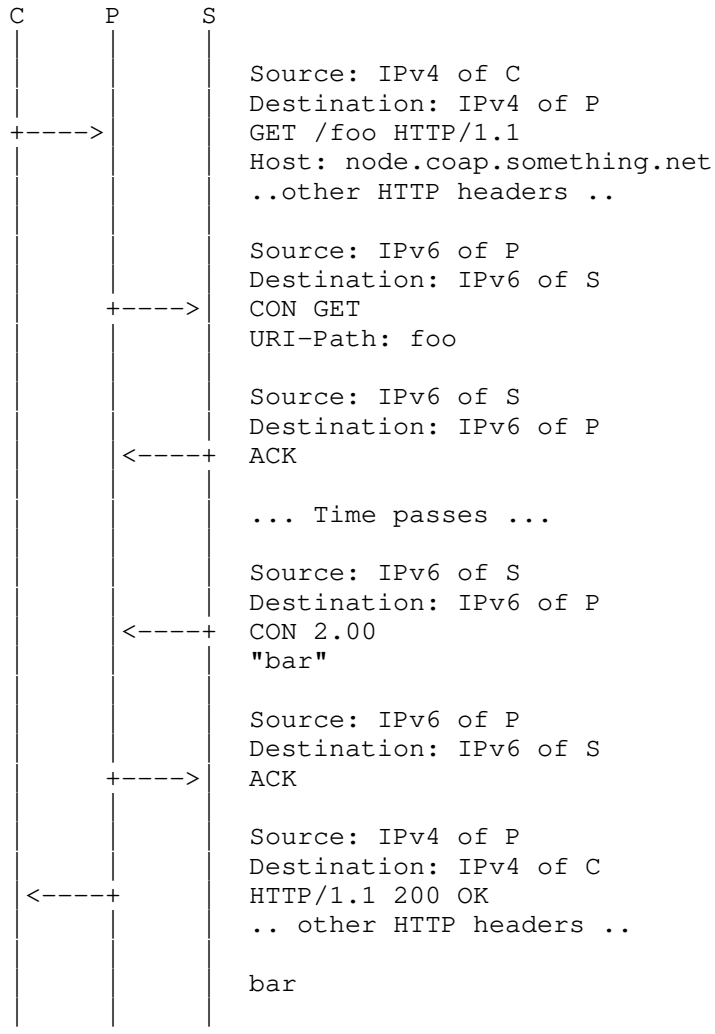


Figure 2: HTTP/IPv4 to CoAP/IPv6 mapping

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. Thus IPv6 connectivity is not required at the HTTP client when accessing a CoAP server over IPv6 only, which is a typically expected use case.

When P is an intercepting HC proxy, the CoAP request SHOULD have the IPv6 address of C as source (IPv4 can always be mapped into IPv6).

When the HTTP client has native IPv6 support, a convenient deployment choice should be to use an HC intecepting proxy. Thus the proxy MUST be located in the IPv6 network path between the client and the server, thus near to the server itself in order to support any Internet client.

2.4. Multiplexing CoAP responses

Defining the mapping of some advanced CoAP features to HTTP (i.e. multicast, observe) must address the need to asynchronously deliver multiple responses to the same HTTP request.

Some HTTP features are useful to succesfully represent these particular sessions.

Using Multipart media type is a suitable solution to deliver multiple CoAP responses within a single HTTP response.

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.

An HC proxy may prefer to transfer each CoAP response immediately after its reception. Responses can be immediately transferred in "chunks" of an HTTP chunked Transfer-Encoding session, without knowing in advance the total number of responses and with arbitrary delay between them.

A detailed discussion on the use of chunked Transfer-Encoding to stream data over HTTP can be found in [I-D.loreto-http-bidirectional]. 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].

When responses are coming from different sources, i.e. multicast, details about the actual source of each CoAP response SHOULD be provided. Source information can be represented in HTTP using a Link option described in [RFC5988] using "via" relation type.

Figure 3 shows an HTTP client (C) requesting the resource "/foo" to a group of CoAP servers (S1/S2/S3) through an HC proxy (P). Discussion related to group communication in CoAP can be found in [I-D.rahman-core-groupcomm].

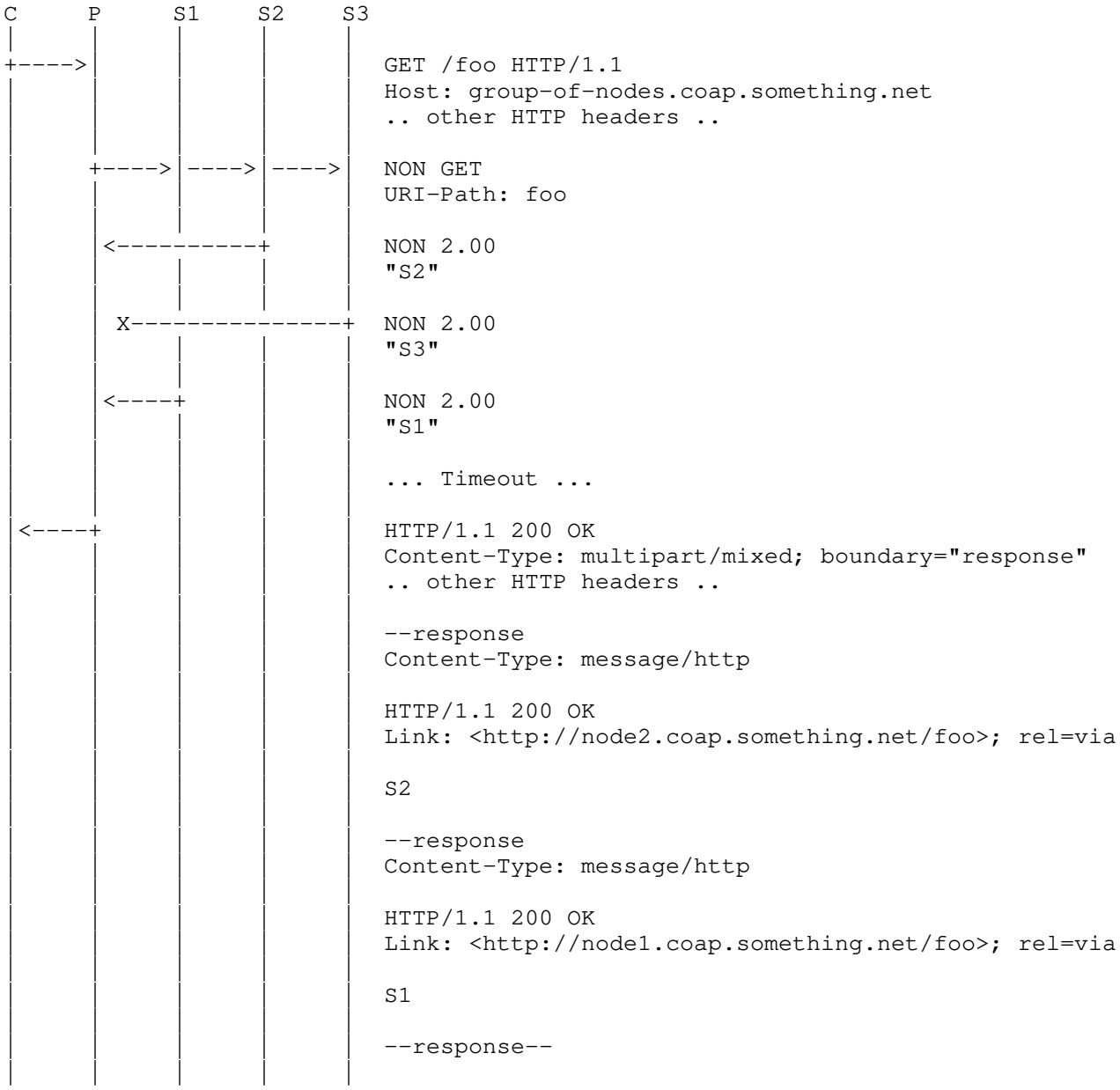


Figure 3: Unicast HTTP to multicast CoAP mapping

The mapping proposed in the above diagram does not make any assumption in how multicasting is done on the constrained network.

If IPv6 multicast support is present in the constrained network, an AAAA record containing the IPv6 multicast group will start multicast operation at the proxy. Otherwise the authority part of the URI is used by the HC proxy to match with a locally defined group of nodes.

In order to minimize the delay in delivering the responses (e.g. HTTP client can incrementally process the responses, HC proxy can reduce internal buffering), each CoAP response can be immediately streamed using HTTP chunked Transfer-Encoding. This encoding was not shown in order to simplify Figure 3, an example showing immediate delivery of CoAP responses is provided in Figure 4 (observe session).

2.4.1. Establishing a CoAP subscription

Using an exchange similar to the one shown in Figure 3, a CoAP observe session can be directly established by a willing HTTP client. Observe mechanism is specified in [I-D.ietf-core-observe].

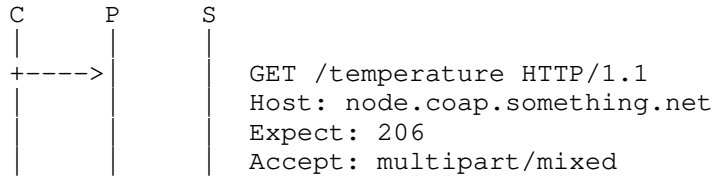
An HTTP client willing to establish a subscription to the "/temperature" resource of a CoAP server SHOULD send an HTTP request with Expect header set to "206" and Accept header set to "multipart/mixed".

The Lifetime of the subscription itself SHALL be sent defining the subscription interval using "Date:" header as starting time and "If-Modified-Since:" as ending time. The HC proxy can compute Lifetime option by using that HTTP headers.

Due to the asynchronous nature of this exchange, the HC proxy willing to accept establishing a subscription SHOULD send an HTTP response with status "206 Partial Content", Content-Type "multipart/mixed" and Transfer-Encoding "chunked".

Each CoAP response will be delivered in a different HTTP chunk until the subscription lifetime expires, when the subscription has expired the HTTP session MUST be closed.

If the HC proxy does not support this exchange or is not willing to establish this session, it SHOULD fail with status "417 Expectation failed".



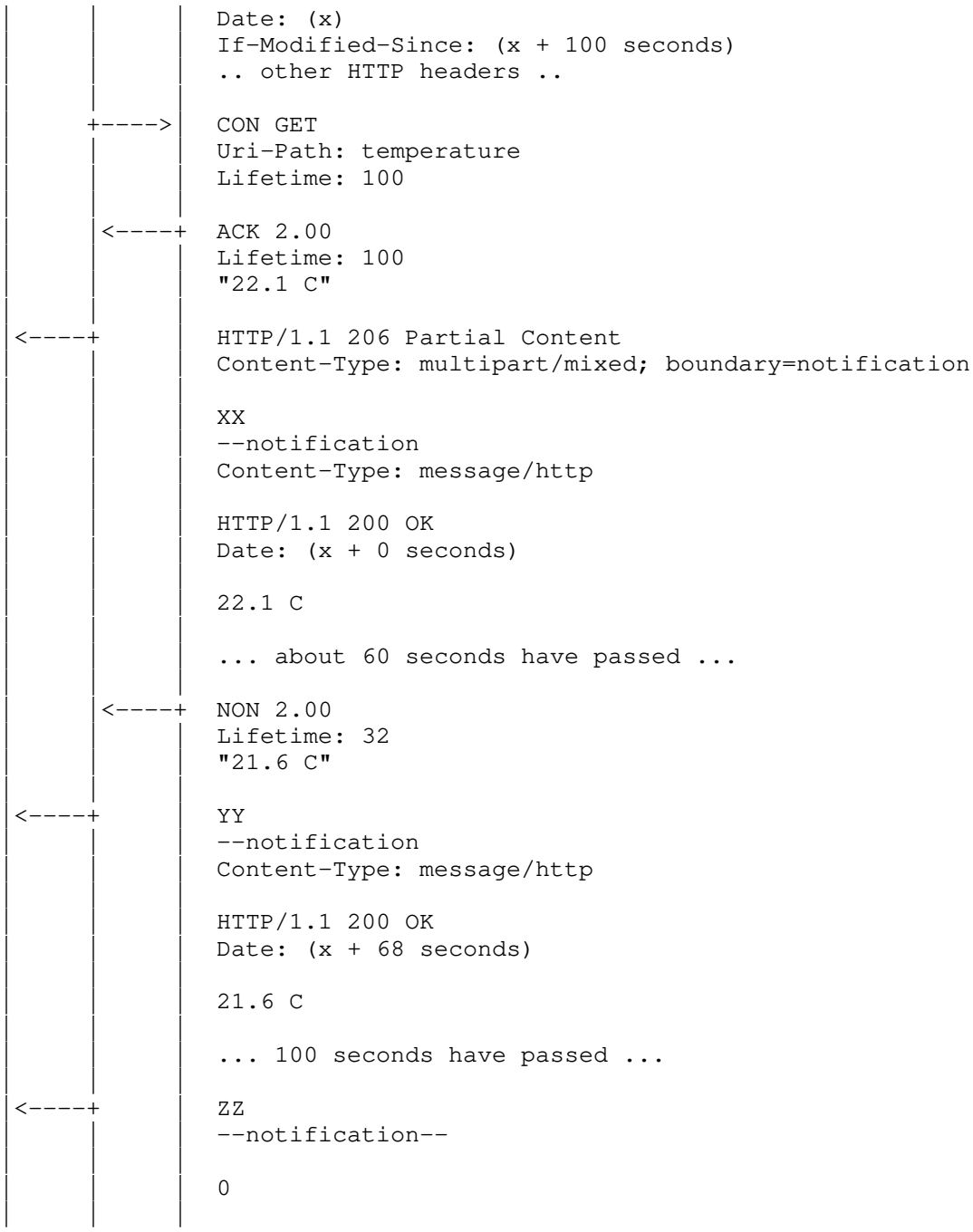

Date: (x)
If-Modified-Since: (x + 100 seconds)
.. other HTTP headers ..
CON GET
Uri-Path: temperature
Lifetime: 100
ACK 2.00
Lifetime: 100
"22.1 C"
HTTP/1.1 206 Partial Content
Content-Type: multipart/mixed; boundary=notification
XX
--notification
Content-Type: message/http
HTTP/1.1 200 OK
Date: (x + 0 seconds)
22.1 C
... about 60 seconds have passed ...
NON 2.00
Lifetime: 32
"21.6 C"
YY
--notification
Content-Type: message/http
HTTP/1.1 200 OK
Date: (x + 68 seconds)
21.6 C
... 100 seconds have passed ...
ZZ
--notification--
0

Figure 4: HTTP subscription to a CoAP resource

When an HTTP client performs direct subscriptions to CoAP servers using this method, the HC proxy has to keep for a possibly long time state information about the observe session and an open HTTP/TCP session to the client.

Soft state required by the various involved protocols (HTTP/TCP, CoAP/UDP) leads to scalability issues when an high number of direct subscriptions are established using the same HC proxy.

Moreover the HC proxy has an active role in the subscription process, thus if crashed or rebooted the subscription to the CoAP node will be lost.

HTTP clients in the real world usually implement notification mechanisms over HTTP using a technique called "Long Polling", an extensive description of this technique is available in Section 2 of [I-D.loreto-http-bidirectional]. A mapping using a "Long Polling" may be identified and can be preferred for longer sessions of observe.

3. CoAP-HTTP

TBD

4. Security Considerations

TBD

5. IANA Considerations

This document does not require any actions by the IANA.

6. Acknowledgements

Special thanks to Nicola Bui and Michele Zorzi for their support and for the various contributions to this document.

Thanks to Kerry Lynn, Akbar Rahman, Peter van der Stok and Anders Brandt for the extensive fruitful discussion about URI mapping, DNS and multicast group communication useful for writing various sections of this document.

Thanks to Brian Frank for its support and its feedback about the content.

7. References

7.1. Normative References

[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.

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

[I-D.ietf-httpbis-p1-messaging] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., and J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-12 (work in progress), October 2010.

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-04 (work in progress), January 2011.

[I-D.ietf-core-observe] Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-01 (work in progress), February 2011.

7.2. Informative References

[I-D.loreto-http-bidirectional] 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", draft-loreto-http-bidirectional-07 (work in progress), January 2011.

[I-D.thomson-hybi-http-timeout] Thomson, M., Loreto, S., and G. Wilkins, "Hypertext Transfer Protocol (HTTP) Timeouts", draft-thomson-hybi-http-timeout-00 (work in progress), March 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.rahman-core-groupcomm]
Rahman, A., "Group Communication for CoAP",
draft-rahman-core-groupcomm-04 (work in progress),
March 2011.

Authors' Addresses

Angelo P. Castellani
University of Padova
Via Gradenigo 6/B
Padova 35131
Italy

Email: angelo@castellani.net

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

CoRE Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: January 9, 2017

C. Bormann
Universitaet Bremen TZI
Z. Shelby, Ed.
ARM

July 08, 2016

Block-wise transfers in CoAP
draft-ietf-core-block-21

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.

A CoAP implementation that does not support these options generally is limited in the size of the representations that can be exchanged. There is therefore an expectation that the Block options are very widely implemented in CoAP implementations, which is why this specification is listed as "updating" RFC 7252.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The work on Constrained RESTful Environments (CoRE) aims at realizing the REST architecture in a suitable form for the most constrained nodes (such as microcontrollers with limited RAM and ROM [RFC7228]) and networks (such as 6LoWPAN, [RFC4944]) [RFC7252]. The CoAP protocol is intended to provide RESTful [REST] services not unlike HTTP [RFC7230], 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:

- reducing implementation complexity in order to minimize code size,
- 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,
- 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) for the transport of larger representations would be 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.

The present 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 the need for 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.)

Block-wise transfers are realized as combinations of exchanges, each of which is performed according to the CoAP base protocol [RFC7252]. Each exchange in such a combination is governed by the specifications in [RFC7252], including the congestion control specifications (Section 4.7 of [RFC7252]) and the security considerations (Section 11 of [RFC7252]; additional security considerations then apply to the transfers as a whole, see Section 7). The present specification minimizes the constraints it adds to those base exchanges; however, not all variants of using CoAP are very useful inside a block-wise transfer (e.g., using Non-confirmable requests within block-wise transfers outside the use case of Section 2.8 would escalate the overall non-delivery probability). To be perfectly clear, the present specification also does not remove any of the constraints posed by the base specification it is strictly layered on top of; e.g., back-to-back packets are limited by Section 4.7 of [RFC7252] (NSTART as a limit for initiating exchanges, PROBING_RATE as a limit for sending with no response): block-wise transfers cannot send/solicit more traffic than a client could be sending to the same server without the block-wise mode.

In some cases, the present specification will RECOMMEND that a client perform a sequence of block-wise transfers "without undue delay". This cannot be phrased as an interoperability requirement, but is an expectation on implementation quality. Conversely, the expectation is that servers will not have go out of their way to accommodate clients that take forever to finish a block-wise transfer. E.g., for a block-wise GET, if the resource changes while this proceeds, the ETag for a further block obtained may be different. To avoid this happening all the time for a fast-changing resource, a server MAY try

to keep a cache around for a specific client for a short amount of time. The expectation here is that the lifetime for such a cache can be kept short, on the order of a few expected round-trip times, counting from the previous block transferred.

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:

- Transfers larger than what can be accommodated in constrained-network link-layer packets can be performed in smaller blocks.
- No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.
- The transfer of each block is acknowledged, enabling individual retransmission if required.
- Both sides have a say in the block size that actually will be used.
- The resulting exchanges are easy to understand using packet analyzer tools and thus quite accessible to debugging.
- If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

A CoAP implementation that does not support these options generally is limited in the size of the representations that can be exchanged, see Section 4.6 of [RFC7252]. Even though the options are Critical, a server may decide to start using them in an unsolicited way in a response. No effort was expended to provide a capability indication mechanism supporting that decision: since the block-wise transfer mechanisms are so fundamental to the use of CoAP for representations larger than about a kilobyte, there is an expectation that they are very widely implemented.

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 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:

- by the maximum datagram size (~ 64 KiB for UDP)
- by the desire to avoid IP fragmentation (MTU of 1280 for IPv6)
- 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 naming these options (for block-wise transfers as well as in Section 4), we use the number 1 ("Block1", "Size1") to refer to the transfer of the resource representation that pertains to the request, and the number 2 ("Block2", "Size2") to refer to the transfer of the resource representation for the response.

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. The Content-Format option applies to the body, not to the payload, in particular the boundaries between the blocks may be in places that are not separating whole units in terms of the structure, encoding, or content-coding used by the Content-Format. (Similarly, the ETag option defined in Section 5.10.6 of [RFC7252] applies to the whole representation of the resource and thus to the body of the response.)

In most cases, all blocks being transferred for a body (except for the last one) will be of the same size. (If the first request uses a bigger block size than the receiver prefers, subsequent requests will use the preferred block 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 Block2 and Block1 Options

No.	C	U	N	R	Name	Format	Length	Default
23	C	U	-	-	Block2	uint	0-3	(none)
27	C	U	-	-	Block1	uint	0-3	(none)

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 [RFC7252], 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 5.5 of [RFC7252]).

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 specific 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.

2.2. Structure of a Block Option

Three items of information may need to be transferred in a Block (Block1 or Block2) option:

- The size of the block (SZX);
- 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 Block Option is a variable-size (0 to 3 byte) unsigned integer (uint, see Section 3.2 of [RFC7252]). This integer value encodes these three fields, see Figure 1. (Due to the CoAP uint encoding rules, when all of NUM, M, and SZX happen to be zero, a zero-byte integer will be sent.)

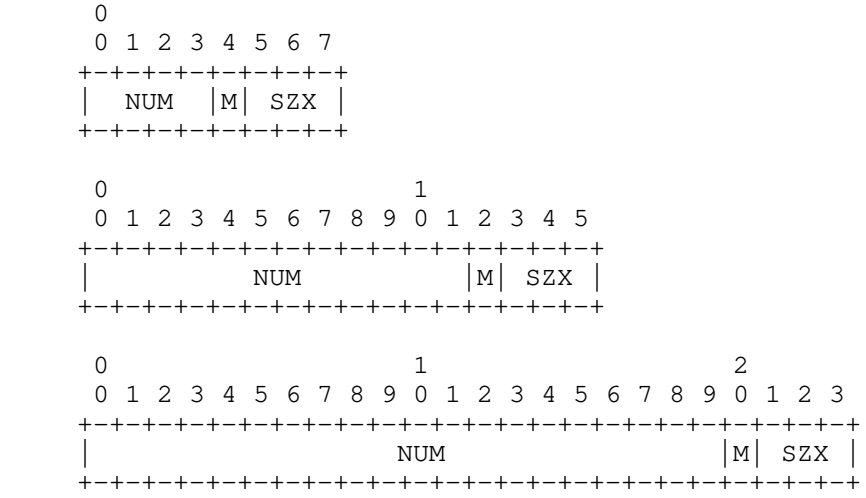


Figure 1: Block option value

The block size is encoded using 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 << (SZX + 4)".

Implementation note: As an implementation convenience, "(val & ~0xF) << (val & 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 first byte of the block being transferred.

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, indicating the block number being requested or provided. Block number 0 indicates the first block of a body (i.e., starting with the first byte of the 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 represented as 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.

There is no default value for the Block1 and Block2 Options. Absence of one of these options is equivalent to an option value of 0 with respect to the value of NUM and M that could be given in the option, i.e. it indicates that the current block is the first and only block of the transfer (block number 0, M bit not set). However, in contrast to the explicit value 0, which would indicate an SZX of 0 and thus a size value of 16 bytes, there is no specific explicit size implied by the absence of the option -- the size is left unspecified. (As for any uint, the explicit value 0 is efficiently indicated by a zero-length option; this, therefore, is different in semantics from the absence of the option.)

2.3. Block Options in Requests and Responses

The Block options are used in one of three roles:

- In descriptive usage, i.e., a Block2 Option in a response (such as a 2.05 response for GET), or a Block1 Option in a request (a 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 need to be transferred to complete the transfer of that body.

  * The block size implied 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 in the response. (The effect is that, if the server uses the smaller of (1) its preferred block size and (2) the block size requested, all blocks for a body use the same block size.)

- 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 a non-zero block number).

- 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 from the same endpoint and plans to implement the request atomically (e.g., acts only upon reception of the last block of payload). In this case, the response MUST NOT carry a Block2 option.

    + 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. (Note that the resource is now in a partially updated state; this approach is only appropriate where exposing such an intermediate state is acceptable. The client can reduce the window by quickly continuing to update the resource, or, in case of failure, restarting the update.)

* 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.

Using one or both Block options, a single REST operation can be split into multiple CoAP message exchanges. As specified in [RFC7252], each of these message exchanges uses their own CoAP Message ID.

The Content-Format Option sent with the requests or responses MUST reflect the content-format of the entire body. If blocks of a response body arrive with different content-format options, it is up to the client how to handle this error (it will typically abort any ongoing block-wise transfer). If blocks of a request arrive at a server with mismatching content-format options, the server MUST NOT assemble them into a single request; this usually leads to a 4.08 (Request Entity Incomplete, Section 2.9.2) error response on the mismatching block.

2.4. Using the Block2 Option

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 as the initial request 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 MAY also use the Block2 Option on the initial request, 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 and a first response has been received with a possibly adjusted block size, all further requests in a single block-wise transfer will ultimately converge on using 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 Block2 option in the response.) The server uses 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 ([RFC7252], Section 5.10.6), 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, 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 server may identify the sequence by the combination of the requesting end-point and the URI being the same in each block-wise request.) 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. Clients that want to retrieve all blocks of a resource SHOULD strive to do so without undue delay. Servers can fully expect to be free to discard any cached state after a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) after the last access to the state, however, there is no requirement to always keep the state for as long.

The Block2 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise response payload transfer (e.g., GET) operations to the same resource. This is rarely a

requirement, but as a workaround, a client may vary the cache key (e.g., by using one of several URIs accessing resources with the same semantics, or by varying a proxy-safe elective option).

2.5. Using the Block1 Option

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 indicated 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 block-wise 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. In this case, all success responses to non-final blocks carry the response code 2.31 (Continue, Section 2.9.1). If not all previous blocks are available at the server at the time of processing the final block, the transfer fails and error code 4.08 (Request Entity Incomplete, Section 2.9.2) MUST be returned. A server MAY also return a 4.08 error code for any (final or non-final) Block1 transfer that is not in sequence; clients that do not have specific mechanisms to handle this case therefore SHOULD always start with block zero and send the following blocks in order.

One reason that a client might encounter a 4.08 error code is that the server has already timed out and discarded the partial request body being assembled. Clients SHOULD strive to send all blocks of a

request without undue delay. Servers can fully expect to be free to discard any partial request body when a period of EXCHANGE_LIFETIME ([RFC7252], Section 4.8.2) has elapsed after the most recent block was transferred; however, there is no requirement on a server to always keep the partial request body for as long.

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 its Block1 option than requested is a hint to try a smaller SZX.)

A block-wise transfer of a request payload that is implemented in a stateless fashion at the server is likely to leave the resource being operated on in an inconsistent state during the time the transfer is still ongoing or when the client does not complete the transfer. This characteristic is closer to that of remote file systems than to that of HTTP, where state is always kept on the server during a transfer. Techniques well known from shared file access (e.g., client-specific temporary resources) can be used to mitigate this difference from HTTP.

The Block1 option provides no way for a single endpoint to perform multiple concurrently proceeding block-wise request payload transfer (e.g., PUT or POST) operations to the same resource. Starting a new block-wise sequence of requests to the same resource (before an old sequence from the same endpoint was finished) simply overwrites the context the server may still be keeping. (This is probably exactly what one wants in this case -- the client may simply have restarted and lost its knowledge of the previous sequence.)

2.6. Combining Block-wise Transfers with the Observe Option

The Observe Option provides a way for a client to be notified about changes over time of a resource [RFC7641]. Resources observed by clients may be larger than can be comfortably processed or transferred in one CoAP message. The following rules apply to the combination of block-wise transfers with notifications.

Observation relationships always apply to an entire resource; the Block2 option does not provide a way to observe a single block of a resource.

As with basic GET transfers, the client can indicate its desired block size in a Block2 Option in the GET request establishing or renewing the observation relationship. If the server supports block-

wise transfers, it SHOULD take note of the block size and apply it as a maximum size to all notifications/responses resulting from the GET request (until the client is removed from the list of observers or the entry in that list is updated by the server receiving a new GET request for the resource from the client).

When sending a 2.05 (Content) notification, the server only sends the first block of the representation. The client retrieves the rest of the representation as if it had caused this first response by a GET request, i.e., by using additional GET requests with Block2 options containing NUM values greater than zero. (This results in the transfer of the entire representation, even if only some of the blocks have changed with respect to a previous notification.)

As with other dynamically changing resources, 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, and the reassembling client MUST compare the ETag options (Section 2.4). Even more so than for the general case of Block2, clients that want to retrieve all blocks of a resource they have been notified about with a first block SHOULD strive to do so without undue delay.

See Section 3.4 for examples.

2.7. Combining Block1 and Block2

In PUT and particularly in POST exchanges, both the request body and the response body may be large enough to require the use of block-wise transfers. First, the Block1 transfer of the request body proceeds as usual. In the exchange of the last slice of this block-wise transfer, the response carries the first slice of the Block2 transfer (NUM is zero). To continue this Block2 transfer, the client continues to send requests similar to the requests in the Block1 phase, but leaves out the Block1 options and includes a Block2 request option with non-zero NUM.

Block2 transfers that retrieve the response body for a request that used Block1 MUST be performed in sequential order.

2.8. Combining Block2 with Multicast

A client can use the Block2 option in a multicast GET request with NUM = 0 to aid in limiting the size of the response.

Similarly, a response to a multicast GET request can use a Block2 option with NUM = 0 if the representation is large, or to further limit the size of the response.

In both cases, the client retrieves any further blocks using unicast exchanges; in the unicast requests, the client SHOULD heed any block size preferences indicated by the server in the response to the multicast request.

Other uses of the Block options in conjunction with multicast messages are for further study.

2.9. Response Codes

Two response codes are defined by this specification beyond those already defined in [RFC7252], and another response code is extended in its meaning.

2.9.1. 2.31 Continue

This new success status code indicates that the transfer of this block of the request body was successful and that the server encourages sending further blocks, but that a final outcome of the whole block-wise request cannot yet be determined. No payload is returned with this response code.

2.9.2. 4.08 Request Entity Incomplete

This new client error status code indicates that the server has not received the blocks of the request body that it needs to proceed. The client has not sent all blocks, not sent them in the order required by the server, or has sent them long enough ago that the server has already discarded them.

(Note that one reason for not having the necessary blocks at hand may be a Content-Format mismatch, see Section 2.3. Implementation note: A server can reject a Block1 transfer with this code when NUM != 0 and a different Content-Format is indicated than expected from the current state of the resource. If it implements the transfer in a stateless fashion, it can match up the Content-Format of the block against that of the existing resource. If it implements the transfer in an atomic fashion, it can match up the block against the partially reassembled piece of representation that is going to replace the state of the resource.)

2.9.3. 4.13 Request Entity Too Large

In [RFC7252], Section 5.9.2.9, the response code 4.13 (Request Entity Too Large) is defined to be like HTTP 413 "Request Entity Too Large". [RFC7252] also recommends that this response SHOULD include a Size1 Option (Section 4) to indicate the maximum size of request entity the

server is able and willing to handle, unless the server is not in a position to make this information available.

The present specification allows the server to return this response code at any time during a Block1 transfer to indicate that it does not currently have the resources to store blocks for a transfer that it would intend to implement in an atomic fashion. It also allows the server to return a 4.13 response to a request that does not employ Block1 as a hint for the client to try sending Block1. Finally, a 4.13 response to a request with a Block1 option (control usage, see Section 2.3) where the response carries a smaller SZX in its Block1 option is a hint to try that smaller SZX.

2.10. Caching Considerations

This specification attempts to leave a variety of implementation strategies open for caches, in particular those in caching proxies. E.g., a cache is free to cache blocks individually, but also could wait to obtain the complete representation before it serves parts of it. Partial caching may be more efficient in a cross-proxy (equivalent to a streaming HTTP proxy). A cached block (partial cached response) can be used in place of a complete response to satisfy a block-wise request that is presented to a cache. Note that different blocks can have different Max-Age values, as they are transferred at different times. A response with a block updates the freshness of the complete representation. Individual blocks can be validated, and validating a single block validates the complete representation. A response with a Block1 Option in control usage with the M bit set invalidates cached responses for the target URI.

A cache or proxy that combines responses (e.g., to split blocks in a request or increase the block size in a response, or a cross-proxy) may need to combine 2.31 and 2.01/2.04 responses; a stateless server may be responding with 2.01 only on the first Block1 block transferred, which dominates any 2.04 responses for later blocks.

If-None-Match only works correctly on Block1 requests with (NUM=0) and MUST NOT be used on Block1 requests with NUM != 0.

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 indicating the kind of Block option (1 or 2) followed by a colon, and then the block number (NUM), more bit (M), and block size exponent (2**(SZX+4)) separated 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.

As in [RFC7252], "MID" is used as an abbreviation of "Message ID".

3.1. Block2 Examples

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 a payload of 128 bytes each, and the third ACK contains a payload between 1 and 128 bytes.

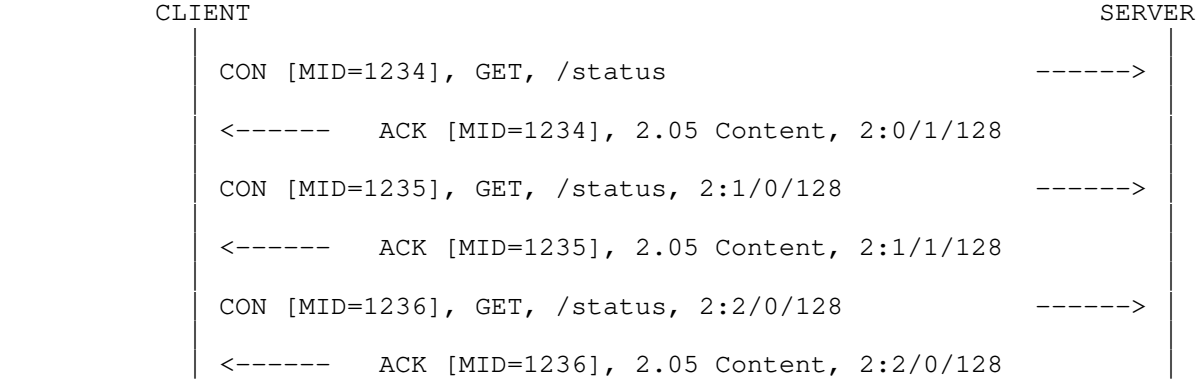


Figure 2: Simple block-wise GET

In the second example (Figure 3), the client anticipates the block-wise transfer (e.g., because of a size indication in the link-format description [RFC6690]) and sends a block size proposal. All ACK messages except for the last carry 64 bytes of payload; the last one carries between 1 and 64 bytes.

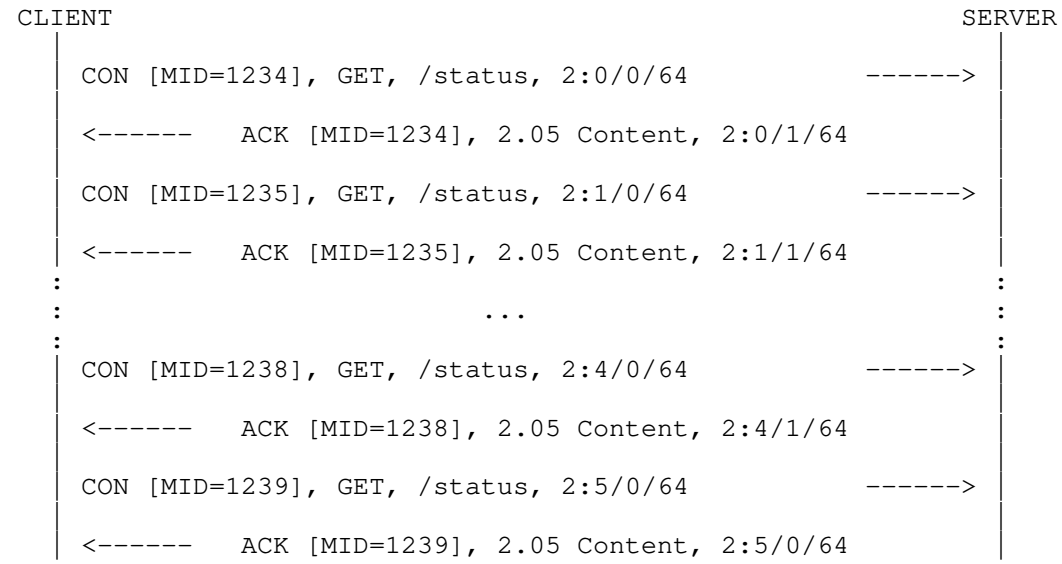


Figure 3: Block-wise GET with early negotiation

In the third example (Figure 4), the client is surprised by the need for a block-wise 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.

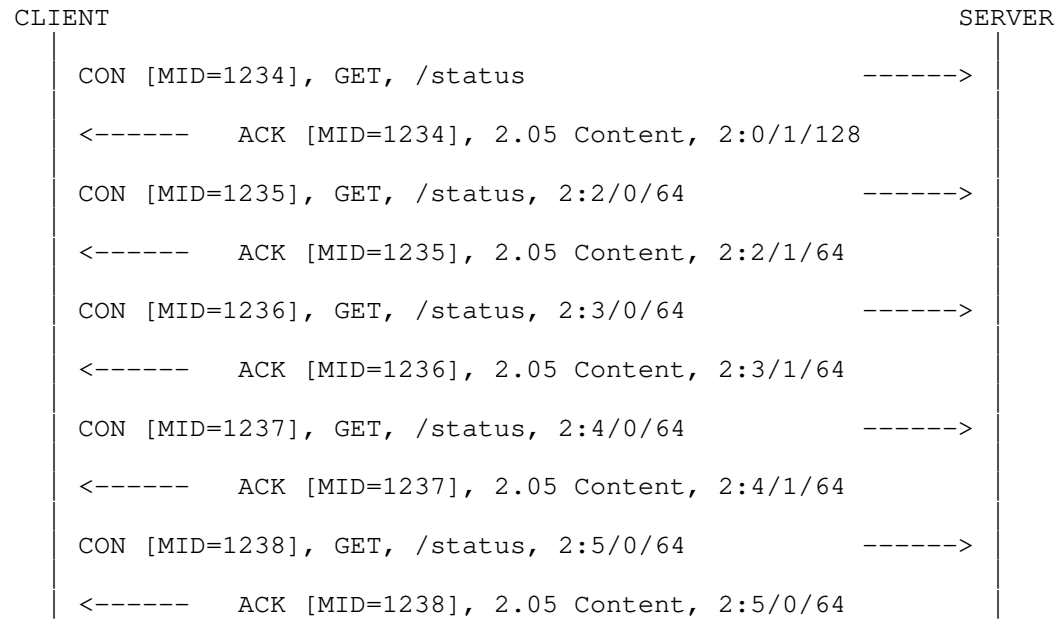


Figure 4: Block-wise GET with late negotiation

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).

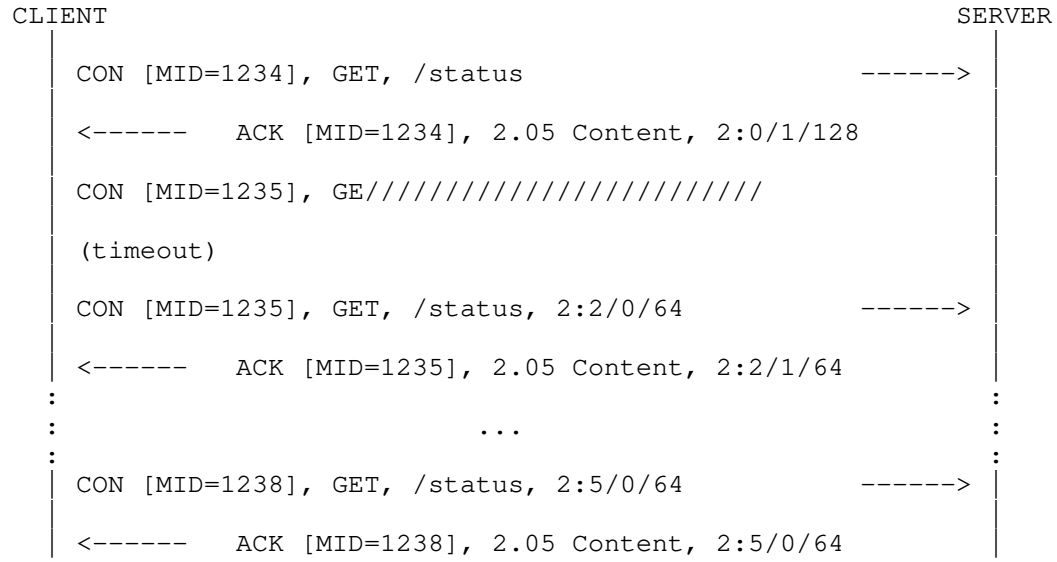


Figure 5: Block-wise GET with late negotiation and lost CON

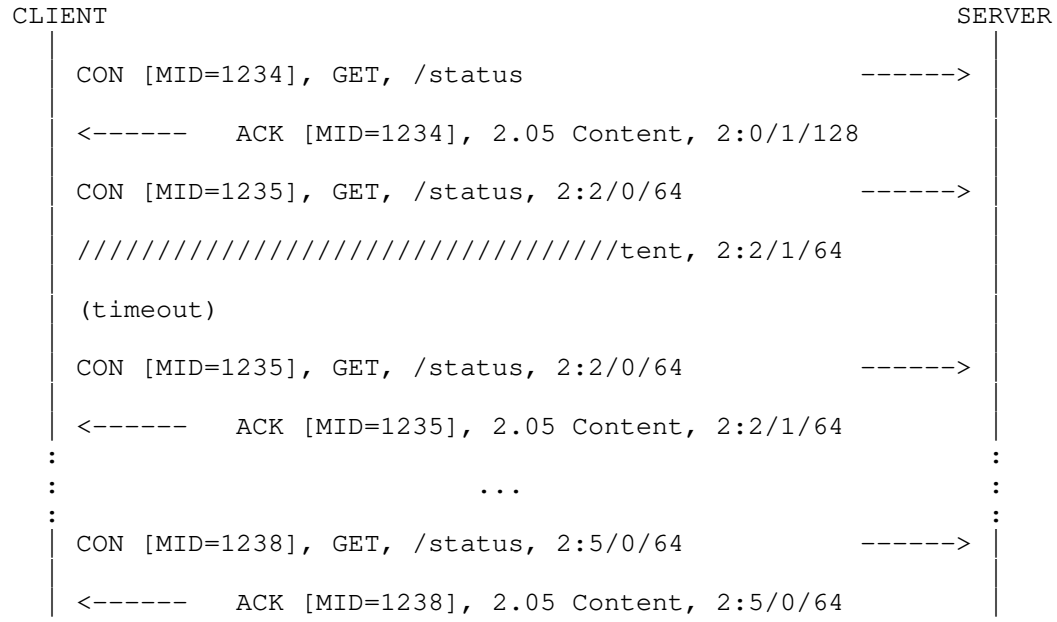


Figure 6: Block-wise GET with late negotiation and lost ACK

3.2. Block1 Examples

The following examples demonstrate a PUT exchange; a POST exchange looks the same, with different requirements on atomicity/idempotence. Note that, similar to GET, the responses to the requests that have a more bit in the request Block1 Option are provisional and carry the response code 2.31 (Continue); only the final response tells the client that the PUT did succeed.

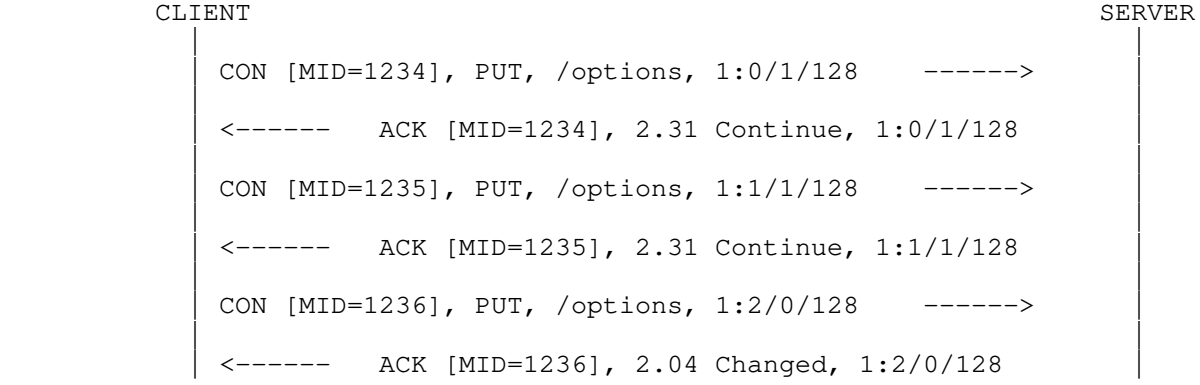


Figure 7: Simple atomic block-wise PUT

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.

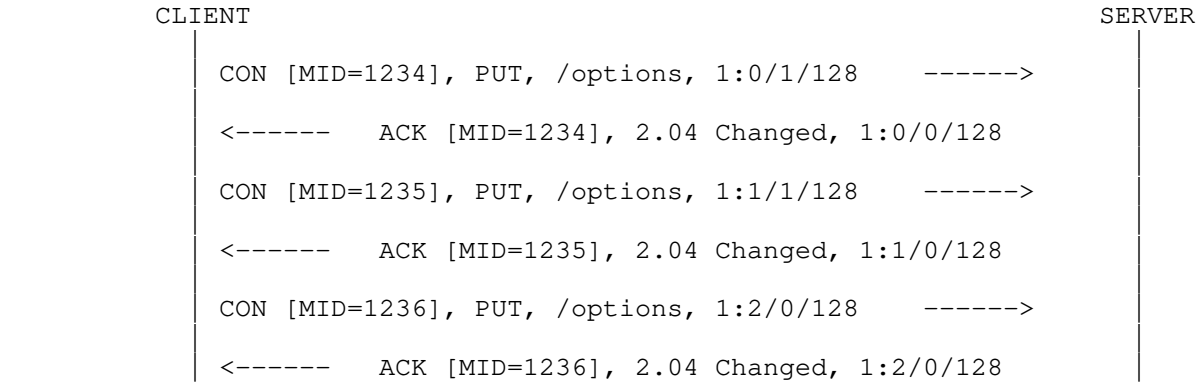


Figure 8: Simple stateless block-wise PUT

Finally, a server receiving a block-wise 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.

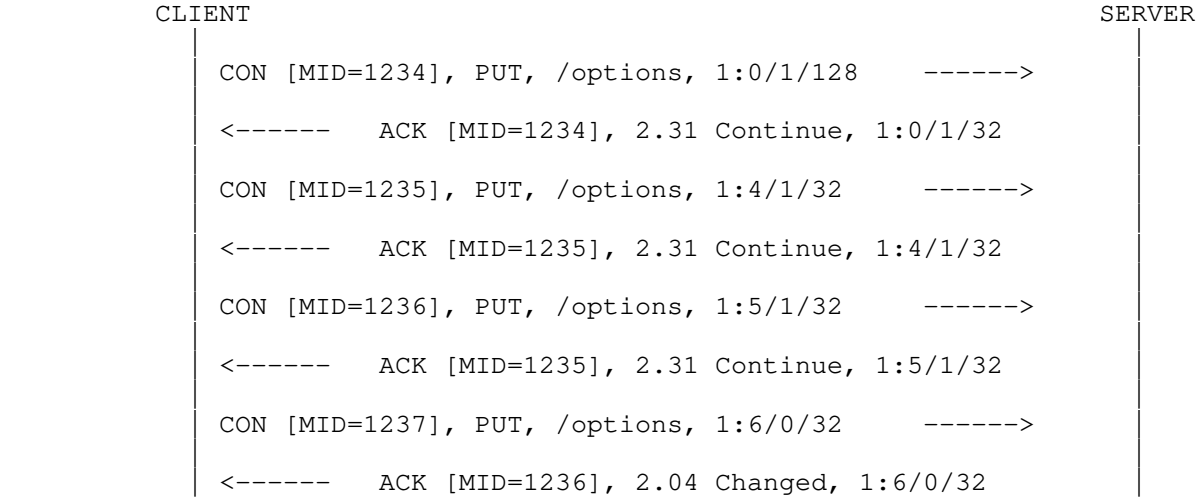


Figure 9: Simple atomic block-wise PUT with negotiation

3.3. Combining Block1 and Block2

Block options may be used in both directions of a single exchange. The following example demonstrates a block-wise POST request, resulting in a separate block-wise response.

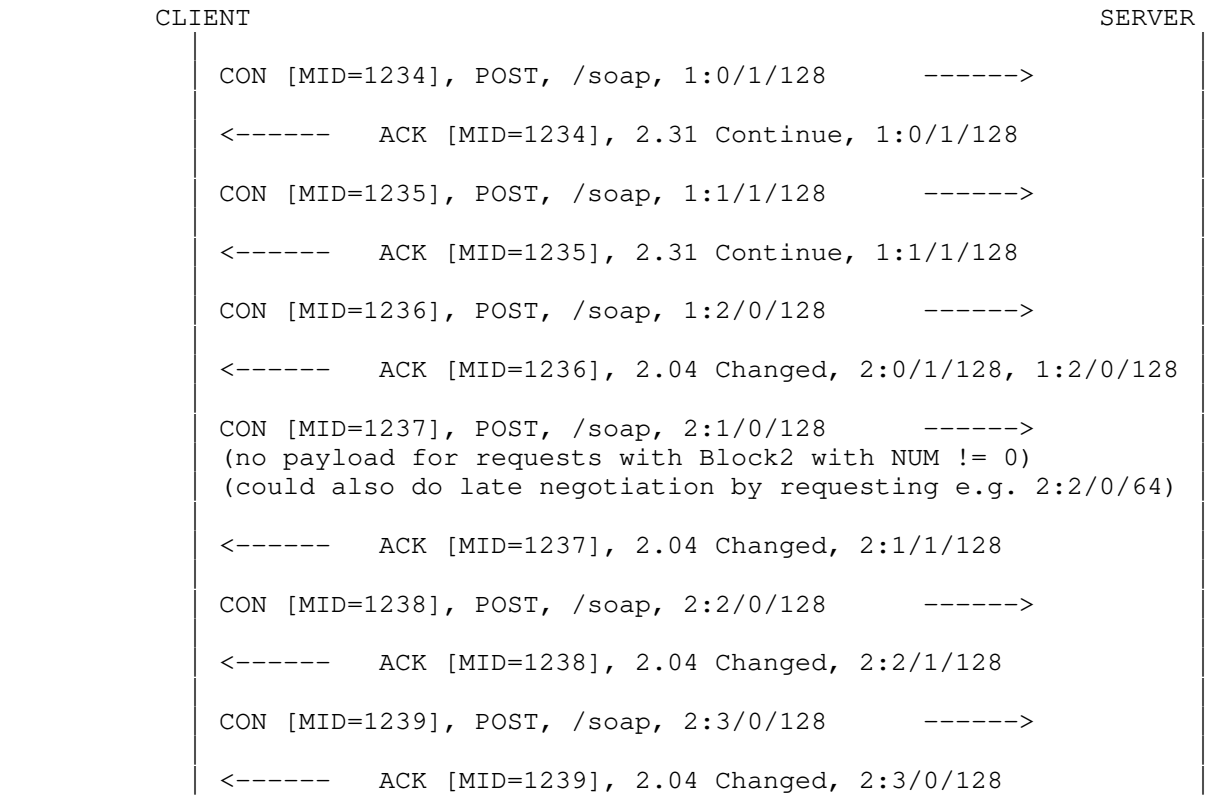


Figure 10: Atomic block-wise POST with block-wise response

This model does provide for early negotiation input to the Block2 block-wise transfer, as shown below.

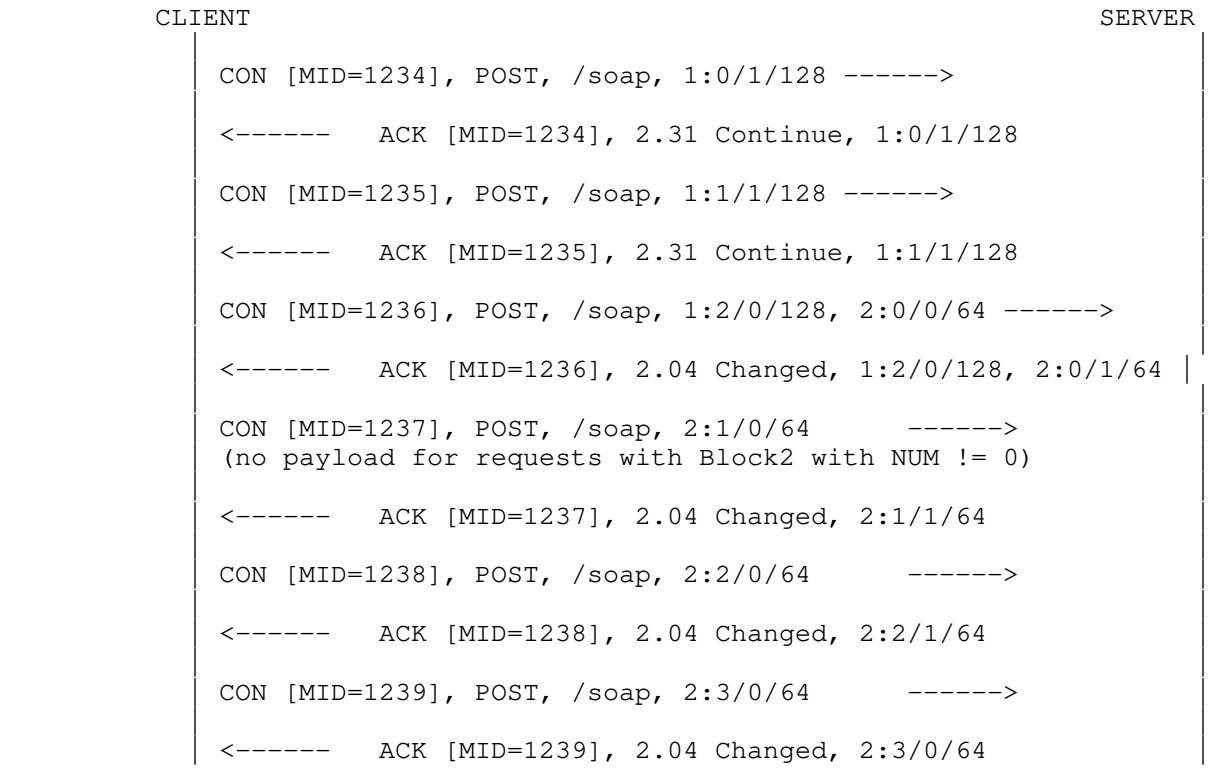
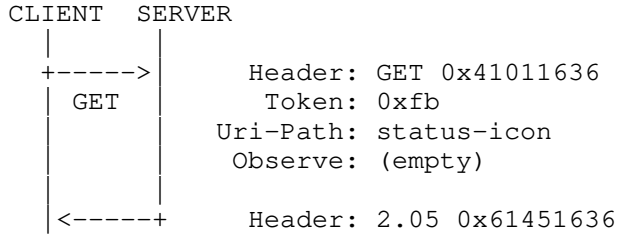


Figure 11: Atomic block-wise POST with block-wise response, early negotiation

3.4. Combining Observe and Block2

In the following example, the server first sends a direct response (Observe sequence number 62350) to the initial GET request (the resulting block-wise transfer is as in Figure 4 and has therefore been left out). The second transfer is started by a 2.05 notification that contains just the first block (Observe sequence number 62354); the client then goes on to obtain the rest of the blocks.



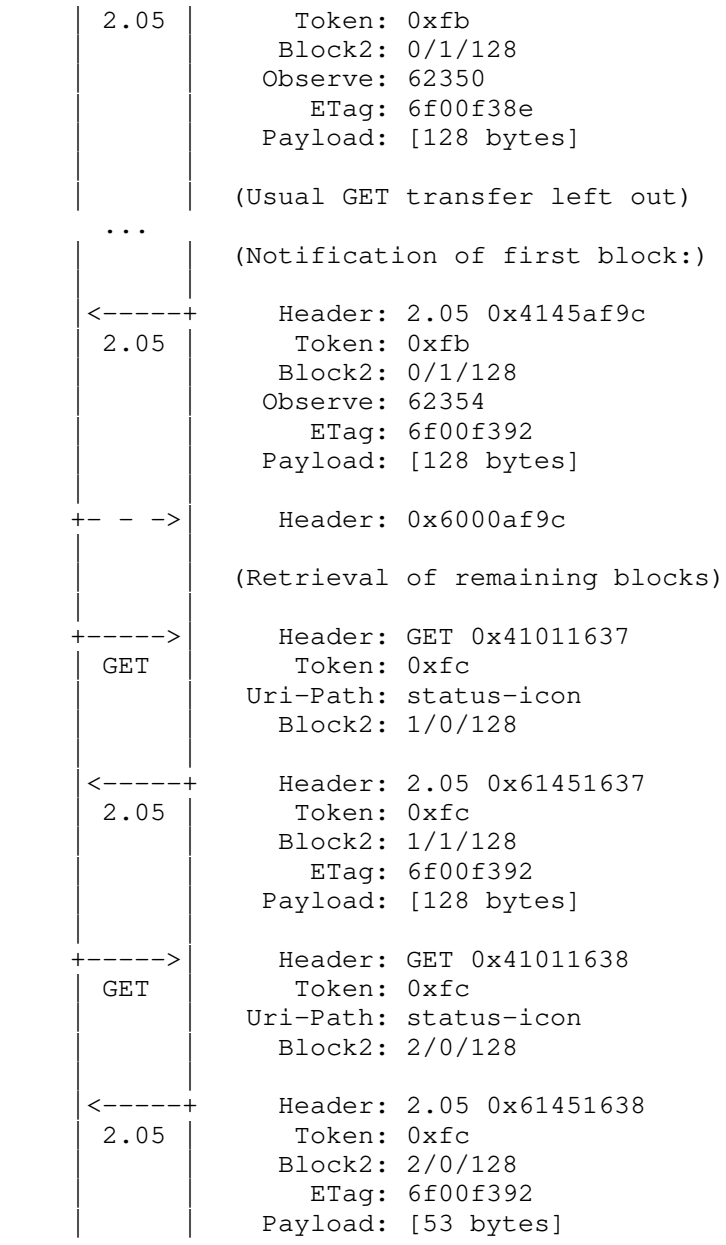
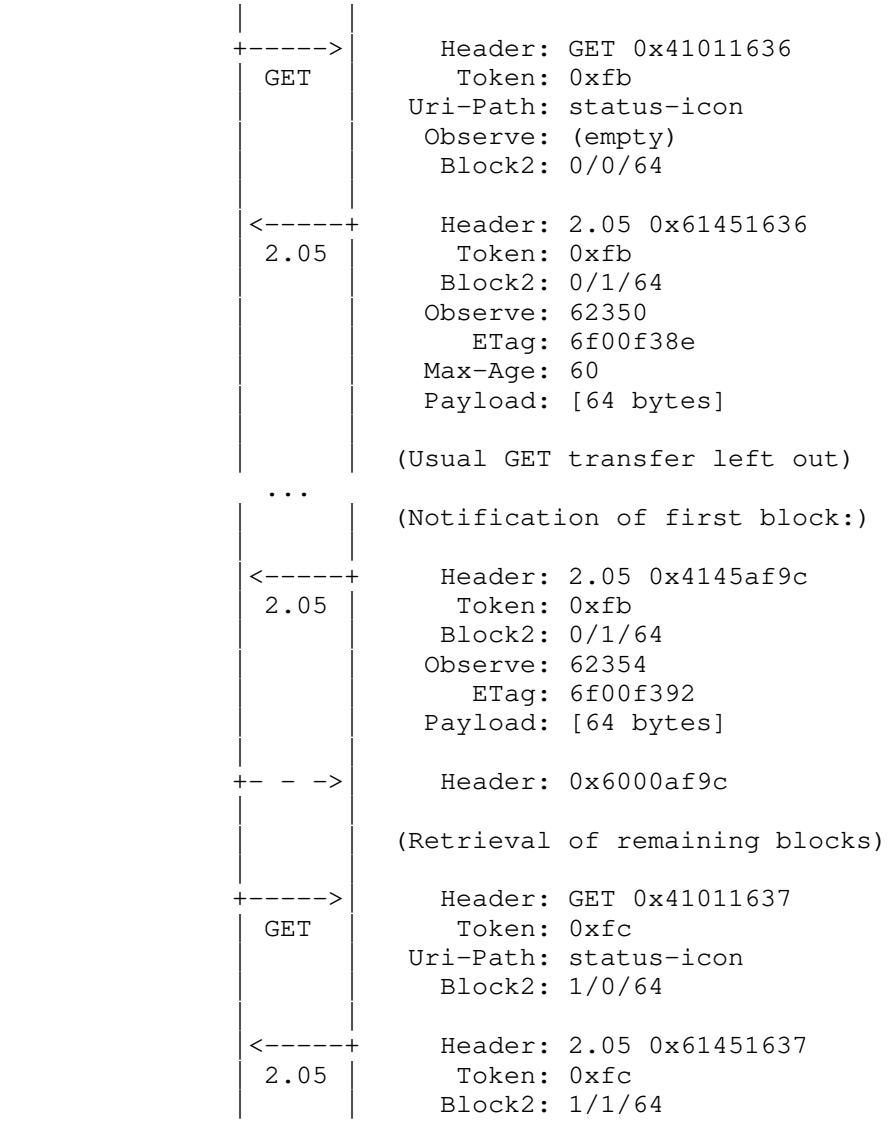


Figure 12: Observe sequence with block-wise response

(Note that the choice of token 0xfc in this examples is arbitrary; tokens are just shown in this example to illustrate that the requests

for additional blocks cannot make use of the token of the Observation relationship. As a general comment on tokens, there is no other mention of tokens in this document, as block-wise transfers handle tokens like any other CoAP exchange. As usual the client is free to choose tokens for each exchange as it likes.)

In the following example, the client also uses early negotiation to limit the block size to 64 bytes.



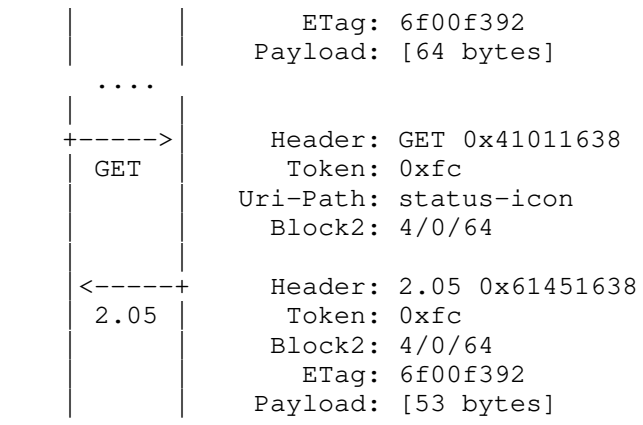


Figure 13: Observe sequence with early negotiation

4. The Size2 and Size1 Options

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 [RFC6690]. However, the size may vary dynamically, so a more up-to-date indication may be useful.

This specification defines two CoAP Options, Size1 for indicating the size of the representation transferred in requests, and Size2 for indicating the size of the representation transferred in responses. (Size1 has already been defined in Section 5.10.9 of [RFC7252] to provide "size information about the resource representation in a request", however that section only details the narrow case of indicating in 4.13 responses the maximum size of request payload that the server is able and willing to handle. The present specification provides details about its use as a request option as well.)

The Size2 Option may be used for two purposes:

- in a request, to ask the server to provide a size estimate along with the usual response ("size request"). For this usage, the value MUST be set to 0.

- in a response carrying a Block2 Option, to indicate the current estimate the server has of the total size of the resource representation, measured in bytes ("size indication").

Similarly, the Size1 Option may be used for two purposes:

- in a request carrying a Block1 Option, to indicate the current estimate the client has of the total size of the resource representation, measured in bytes ("size indication").

- in a 4.13 response, to indicate the maximum size that would have been acceptable [RFC7252], measured in bytes.

Apart from conveying/asking for size information, the Size options have no other effect on the processing of the request or response. If the client wants to minimize the size of the payload in the resulting response, it should add a Block2 option to the request with a small block size (e.g., setting SZX=0).

The Size Options are "elective", i.e., a client MUST be prepared for the server to ignore the size estimate request. The Size Options MUST NOT occur more than once.

No.	C	U	N	R	Name	Format	Length	Default
60			x		Size1	uint	0-4	(none)
28			x		Size2	uint	0-4	(none)

Table 2: Size Option Numbers

Implementation Notes:

- As a quality of implementation consideration, block-wise 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.

- The end of a block-wise transfer is governed by the M bits in the Block Options, _not_ by exhausting the size estimates exchanged.

- 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 for either Size Option.

- The Size Options are neither critical nor unsafe, and are marked as No-Cache-Key.

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, historically there was more variation in how HTTP servers might implement ranges; recently, [RFC7233] has defined that Range header fields received with a request method other than GET are not to be interpreted. So, in general, the CoAP-to-HTTP intermediary will have to try sending the payload of all the blocks of a block-wise transfer for these other methods 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 [RFC7252]:

Number	Name	Reference
23	Block2	[RFCXXXX]
27	Block1	[RFCXXXX]
28	Size2	[RFCXXXX]

Table 3: CoAP Option Numbers

This draft adds the following response code to the CoAP Response Codes registry of [RFC7252]:

Code	Description	Reference
2.31	Continue	[RFCXXXX]
4.08	Request Entity Incomplete	[RFCXXXX]

Table 4: 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 specific security associations, 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. Future end-to-end security mechanisms that may be added to CoAP itself may have related security considerations, this includes considerations about caching of blocks in clients and in proxies (see Section 2.10 and Section 5 for different strategies in performing this caching); these security considerations will need to be described in the specifications of those mechanisms.

A stateless server might be susceptible to an attack where the adversary sends a Block1 (e.g., PUT) block with a high block number: A naive implementation might exhaust its resources by creating a huge resource representation.

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 block-wise requests may induce the server to create state, e.g. to create a snapshot for the block-wise 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), for instance 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 block-wise transfers in an expedient way to minimize the likelihood of running into such a timeout.

7.2. Mitigating Amplification Attacks

[RFC7252] 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. References

8.1. Normative References

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

[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>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

8.2. Informative References

[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>.

[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, DOI 10.17487/RFC4919, August 2007, <http://www.rfc-editor.org/info/rfc4919>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <http://www.rfc-editor.org/info/rfc4944>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <http://www.rfc-editor.org/info/rfc7233>.

Acknowledgements

Much of the content of this draft is the result of discussions with the [RFC7252] authors, and via many CoRE WG discussions.

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, a solution to the 4.13 ambiguity problem, and the section about combining Block and multicast. Markus Becker proposed getting rid of an ill-conceived default value for the Block2 and Block1 options. Peter Bigot

insisted on a more systematic coverage of the options and response code. Qin Wu provided a review for the IETF Operational directorate, and Goeran Selander commented on the security considerations.

Kepeng Li, Linyi Tian, and Barry Leiba wrote up an early version of the Size Option, which has informed this draft. Klaus Hartke wrote some of the text describing the interaction of Block2 with Observe. Matthias Kovatsch provided a number of significant simplifications of the protocol.

The IESG reviewers provided very useful comments. Spencer Dawkins even suggested new text. Mirja Kuehlewind and he insisted on being more explicit about the layering of block-wise transfers on top of the base protocol. Ben Campbell helped untangling some MUST/SHOULD soup. Comments by Alexey Melnikov, as well as the gen-art review by Jouni Korhonen and the ops-dir review by Qin Wu, caused further improvements to the text.

Authors' Addresses

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

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

Zach Shelby (editor)
ARM
150 Rose Orchard
San Jose, CA 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2013

Z. Shelby
Sensinode
K. Hartke
C. Bormann
Universitaet Bremen TZI
June 28, 2013

Constrained Application Protocol (CoAP)
draft-ietf-core-coap-18

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 endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

Table of Contents

1. Introduction

The use of web services (web APIs) 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 such as 6LoWPAN support the fragmentation of IPv6 packets into small link-layer frames, however incurring significant reduction in packet delivery probability. One design goal of CoAP has been to keep message overhead small, thus limiting the need for 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 refashioning simple HTTP interfaces into a more compact protocol, 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:

- Constrained web protocol fulfilling M2M requirements.
- UDP [RFC0768] binding with optional reliability supporting unicast and multicast requests.
- Asynchronous message exchanges.
- Low header overhead and parsing complexity.
- URI and Content-type support.

- Simple proxy and caching capabilities.

- 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.

- Security binding to Datagram Transport Layer Security (DTLS) [RFC6347].

1.2. 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. 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], including "resource", "representation", "cache", and "fresh". 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. A common form of an intermediary is a proxy; several classes of such proxies are discussed in this specification.

Proxy
: An intermediary that mainly is concerned with forwarding requests and relaying back responses, possibly performing caching, namespace translation, or protocol translation in the process. As opposed to intermediaries in the general sense, proxies generally do not implement specific application semantics. Based on the position in the overall structure of the request forwarding, there are two common forms of proxy: forward-proxy and reverse-proxy. In some cases, a single endpoint might act as an origin server, forward-proxy, or reverse-proxy, switching behavior based on the nature of each request.

Forward-Proxy
: A "forward-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 stands in for one or more other server(s) and satisfies requests on behalf of these, doing any necessary translations. Unlike a forward-proxy, the client may not be aware that it is communicating with a reverse-proxy; a reverse-proxy receives requests as if it was the origin server for the target resource.

CoAP-to-CoAP Proxy
: A proxy that maps from a CoAP request to a CoAP request, i.e. uses the CoAP protocol both on the server and the client side. Contrast to cross-proxy.

Cross-Proxy
: A cross-protocol proxy, or "cross-proxy" for short, is a proxy that translates between different protocols, such as a CoAP-to-HTTP proxy or an HTTP-to-CoAP proxy. While this specification makes very specific demands of CoAP-to-CoAP proxies, there is more variation possible in cross-proxies.

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.

Acknowledgement Message
: An Acknowledgement message acknowledges that a specific Confirmable message arrived. By itself, an Acknowledgement message does not indicate success or failure of any request encapsulated in the Confirmable message, but the Acknowledgement message may also carry a Piggy-Backed Response (q.v.).

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. Provoking a Reset message (e.g., by sending an Empty Confirmable message) is also useful as an inexpensive check of the liveness of an endpoint ("CoAP ping").

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).

Empty Message
: A message with a Code of 0.00; neither a request nor a response. An Empty message only contains the four-byte header.

Critical Option
: An option that would need to be understood by the endpoint ultimately 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 an error response or summary 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).

Unsafe Option
: An option that would need to be understood by a proxy receiving the message in order to safely forward the message (Section 5.4.2). Not every critical option is an unsafe option.

Safe-to-Forward Option
: An option that is intended to be safe for forwarding by a proxy that does not understand it. Forwarding the message even without understanding the option is acceptable (Section 5.4.2).

Resource Discovery
: The process where a CoAP client queries a server for its list of hosted resources (i.e., links, Section 7).

Content-Format
: The combination of an Internet media type, potentially with specific parameters given, and a content-coding (which is often the identity content-coding), identified by a numeric identifier defined by the CoAP Content-Format Registry. When the focus is less on the numeric identifier than on the combination of these characteristics of a resource representation, this is also called "representation format".

Additional terminology for constrained nodes and constrained node networks can be found in [I-D.ietf-lwig-terminology].

In this specification, the term "byte" is used in its now customary sense as a synonym for "octet".

All multi-byte integers in this protocol are interpreted in network byte order.

Where arithmetic is used, this specification uses the notation familiar from the programming language C, except that 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 Acknowledgement messages.

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.

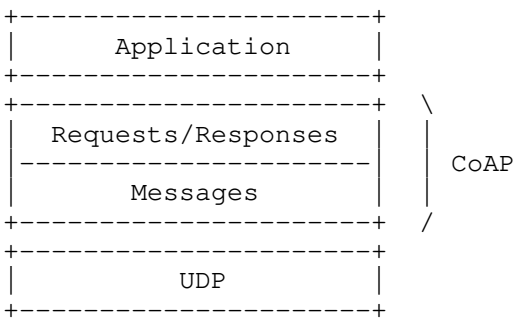


Figure 1: Abstract layering of CoAP

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. (The Message ID is compact; its 16-bit size enables up to about 250 messages per second from one endpoint to another with default protocol parameters.)

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 (in this example, 0x7d34) from the corresponding endpoint; see Figure 2. When a recipient is not at all able to process a Confirmable message (i.e., not even able to provide a suitable error response), it replies with a Reset message (RST) instead of an Acknowledgement (ACK).

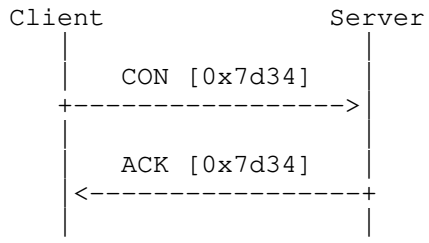


Figure 2: Reliable message transmission

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 (in this example, 0x01a0); see Figure 3. When a recipient is not able to process a Non-confirmable message, it may reply with a Reset message (RST).

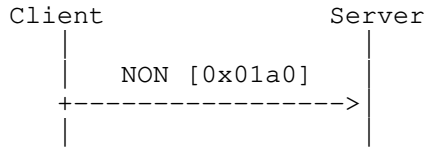


Figure 3: Unreliable message transmission

See Section 4 for details of CoAP messages.

As CoAP runs over 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. This document specifies a binding to DTLS for securing the protocol; the use of IPsec with CoAP is discussed in [I-D.bormann-core-ipsec-for-coap].

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 media type are carried as CoAP options. A Token is used to match responses to requests independently from the underlying messages (Section 5.3). (Note that the Token is a concept separate from the Message ID.)

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. (There is no need for separately acknowledging a piggy-backed response, as the client will retransmit the request if the Acknowledgement message carrying the piggy-backed response is lost.) 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.

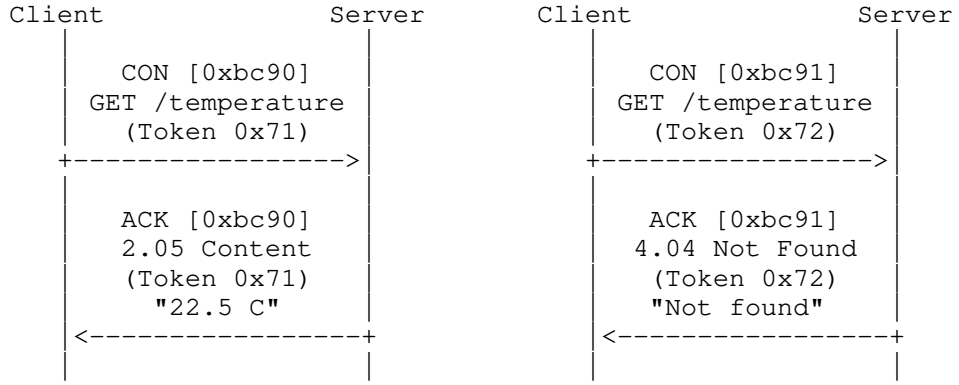


Figure 4: Two GET requests with piggy-backed responses

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.

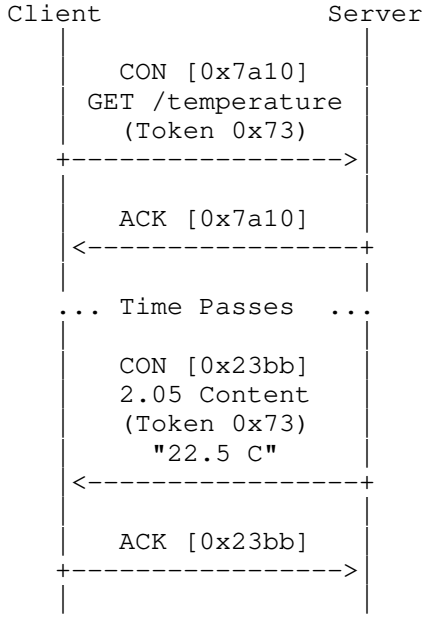
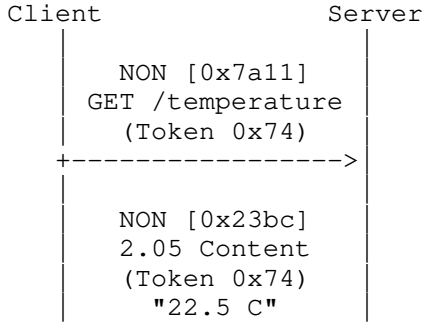


Figure 5: A GET request with a separate response

If a request is sent in a Non-confirmable message, then the response is sent using a new Non-confirmable message, although the server may instead send a Confirmable message. This type of exchange is illustrated in Figure 6.



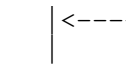


Figure 6: A NON request and response

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" [HHGTTG] 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.)

Methods beyond the basic four can be added to CoAP in separate specifications. New methods do not necessarily have to use requests and responses in pairs. Even for existing methods, a single request may yield multiple responses, e.g. for a multicast request (Section 8) or with the Observe option [I-D.ietf-core-observe].

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 relate 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 [REST] 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 cross-protocol proxy ("cross-proxy"), which

converts the method or response code, media 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 [RFC6690] as discussed in Section 7.

3. Message Format

CoAP is based on the exchange of compact messages which, by default, are transported over UDP (i.e. each CoAP message occupies the data section of one UDP datagram). CoAP may also be used over Datagram Transport Layer Security (DTLS) (see Section 9.1). It could also be used over other transports such as SMS, TCP or SCTP, the specification of which is out of this document's scope. (UDP-lite [RFC3828] and UDP zero checksum [RFC6936] are not supported by CoAP.)

CoAP messages are encoded in a simple binary format. The message format starts with a fixed-size 4-byte header. This is followed by a variable-length Token value which can be between 0 and 8 bytes long. Following the Token value comes a sequence of zero or more CoAP Options in Type-Length-Value (TLV) format, optionally followed by a payload which takes up the rest of the datagram.

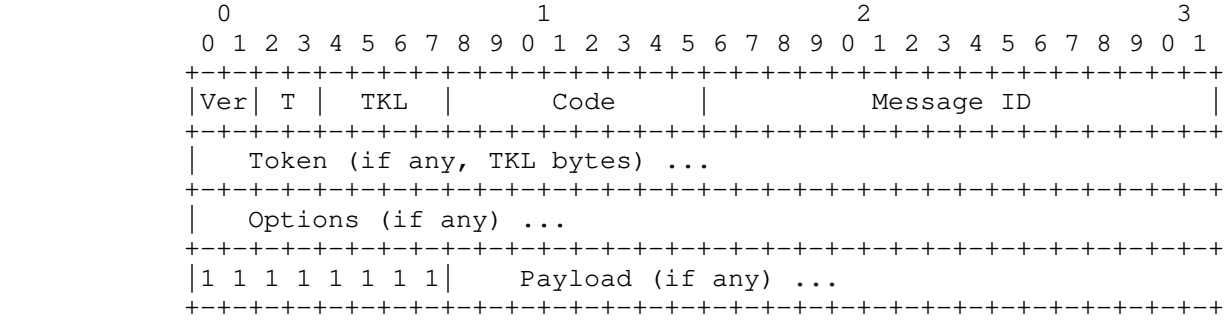


Figure 7: Message Format

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 (01 binary). Other values are reserved for future versions. Messages with unknown version numbers MUST be silently ignored.

Type (T): 2-bit unsigned integer. Indicates if this message is of type Confirmable (0), Non-confirmable (1), Acknowledgement (2) or Reset (3). The semantics of these message types are defined in Section 4.

Token Length (TKL): 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes). Lengths 9-15 are reserved, MUST NOT be sent, and MUST be processed as a message format error.

Code: 8-bit unsigned integer, split into a 3-bit class (most significant bits) and a 5-bit detail (least significant bits), documented as c.dd where c is a digit from 0 to 7 for the 3-bit subfield and dd are two digits from 00 to 31 for the 5-bit subfield. The class can indicate a request (0), a success response (2), a client error response (4), or a server error response (5). (All other class values are reserved.) As a special case, Code 0.00 indicates an Empty message. 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 Registries (Section 12.1). The semantics of requests and responses are defined in Section 5.

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 to messages of type Confirmable/Non-confirmable. The rules for generating a Message ID and matching messages are defined in Section 4.

The header is followed by the Token value, which may be 0 to 8 bytes, as given by the Token Length field. The Token value is used to correlate requests and responses. The rules for generating a Token and correlating requests and responses are defined in Section 5.3.1.

Header and Token are followed by zero or more Options (Section 3.1). An Option can be followed by the end of the message, by another Option, or by the Payload Marker and the payload.

Following the header, token, and options, if any, comes the optional payload. If present and of non-zero length, it is prefixed by a fixed, one-byte Payload Marker (0xFF) which indicates the end of options and the start of the payload. The payload data extends from after the marker to the end of the UDP datagram, i.e., the Payload Length is calculated from the datagram size. The absence of the Payload Marker denotes a zero-length payload. The presence of a marker followed by a zero-length payload MUST be processed as a message format error.

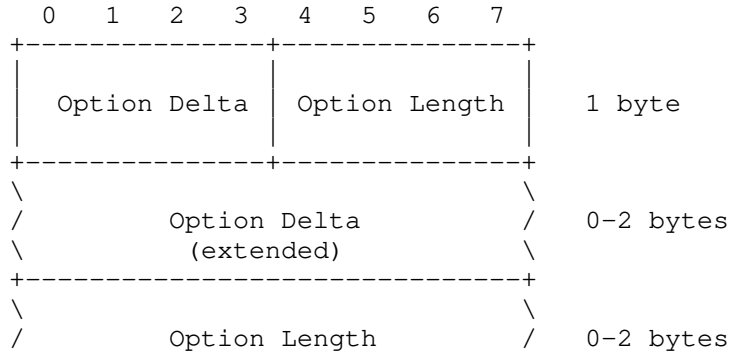
Implementation Note: The byte value 0xFF may also occur within an option length or value, so simple byte-wise scanning for 0xFF is not a viable technique for finding the payload marker. The byte 0xFF has the meaning of a payload marker only where the beginning of another option could occur.

3.1. Option Format

CoAP defines a number of options which can be included in a message. Each option instance in a message specifies the Option Number of the defined CoAP option, the length of the Option Value and the Option Value itself.

Instead of specifying the Option Number directly, the instances MUST appear in order of their Option Numbers and a delta encoding is used between them: The Option Number for each instance is calculated as the sum of its delta and the Option Number of the preceding instance in the message. For the first instance in a message, a preceding option instance with Option Number zero is assumed. Multiple instances of the same option can be included by using a delta of zero.

Option Numbers are maintained in the CoAP Option Number Registry (Section 12.2). See Section 5.4 for the semantics of the options defined in this document.



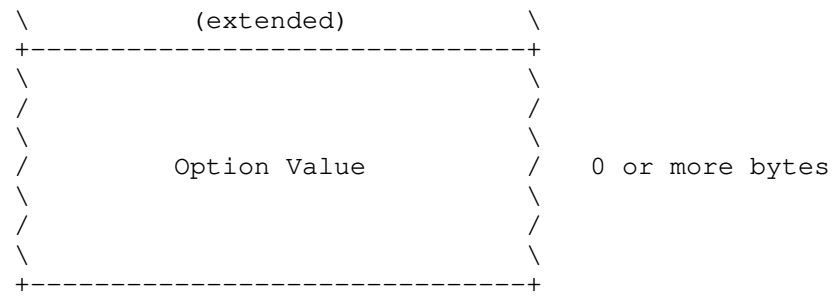


Figure 8: Option Format

The fields in an option are defined as follows:

Option Delta: 4-bit unsigned integer. A value between 0 and 12 indicates the Option Delta. Three values are reserved for special constructs:

13: An 8-bit unsigned integer follows the initial byte and indicates the Option Delta minus 13.

14: A 16-bit unsigned integer in network byte order follows the initial byte and indicates the Option Delta minus 269.

15: Reserved for the Payload Marker. If the field is set to this value but the entire byte is not the payload marker, this MUST be processed as a message format error.

The resulting Option Delta is used as the difference between the Option Number of this option and that of the previous option (or zero for the first option). In other words, the Option Number is calculated by simply summing the Option Delta values of this and all previous options before it.

Option Length: 4-bit unsigned integer. A value between 0 and 12 indicates the length of the Option Value, in bytes. Three values are reserved for special constructs:

13: An 8-bit unsigned integer precedes the Option Value and indicates the Option Length minus 13.

14: A 16-bit unsigned integer in network byte order precedes the Option Value and indicates the Option Length minus 269.

15: Reserved for future use. If the field is set to this value, it MUST be processed as a message format error.

Value: A sequence of exactly Option Length bytes. The length and format of the Option Value depend on the respective option, which MAY define variable length values. See Section 3.2 for the formats used in this document; options defined in other documents MAY make use of other option value formats.

3.2. Option Value Formats

The options defined in this document make use of the following option value formats.

empty: A zero-length sequence of bytes.

opaque: An opaque sequence of bytes.

uint: A non-negative integer which is represented in network byte order using the number of bytes given by the Option Length field.

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 zero bytes. For example, the number 0 is represented with an empty option value (a zero-length sequence of bytes), and the number 1 by a single byte with the numerical value of 1 (bit combination 00000001 in most significant bit first notation). A recipient MUST be prepared to process values with leading zero bytes.

Implementation Note: The exceptional behavior permitted for the sender is intended for highly constrained, templated implementations (e.g., hardware implementations) that use fixed size options in the templates.

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 (except where 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.

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:

- Simple stop-and-wait retransmission reliability with exponential back-off for Confirmable messages.

- 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. The specific definition of an endpoint depends on the transport being used for CoAP. For the transports defined in this specification, the endpoint 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 Type 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 Request/Response Code field in the CoAP Header and is relevant to the request/response model. Possible values for the field are maintained in the CoAP Code Registries (Section 12.1).

An Empty message has the Code field set to 0.00. The Token Length field MUST be set to 0 and bytes of data MUST NOT be present after the Message ID field. If there are any bytes, they MUST be processed as a message format error.

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, unless it is used only to elicit a Reset message in which case it is Empty. A recipient

MUST acknowledge a Confirmable message with an Acknowledgement message or, if it lacks context to process the message properly (including the case where the message is Empty, uses a code with a reserved class (1, 6 or 7), or has a message format error), MUST reject it; rejecting a Confirmable message is effected by sending a matching Reset message and otherwise ignoring it. 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. Rejecting an Acknowledgement or Reset message (including the case where the Acknowledgement carries a request or a code with a reserved class, or the Reset message is not Empty) is effected by silently ignoring it. More generally, recipients of Acknowledgement and Reset messages MUST NOT respond with either Acknowledgement or Reset messages.

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 duration (often not an integral number of seconds) 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 in time, transmission is considered successful.

This specification makes no strong requirements on the accuracy of the clocks used to implement the above binary exponential backoff algorithm. In particular, an endpoint may be late for a specific retransmission due to its sleep schedule, and maybe catch up on the next one. However, the minimum spacing before another retransmission is ACK_TIMEOUT, and the entire sequence of (re-)transmissions MUST stay in the envelope of MAX_TRANSMIT_SPAN (see Section 4.8.2), even if that means a sender may miss an opportunity to transmit.

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 MUST retain the state to create the ACK for the request, if needed, even if a Confirmable response was already acknowledged by the requester.

Another reason for giving up retransmission MAY be the receipt of ICMP errors. If it is desired to take account of ICMP errors, to mitigate potential spoofing attacks, implementations SHOULD take care to check the information about the original datagram in the ICMP message, including port numbers and CoAP header information such as message type and code, Message ID, and Token; if this is not possible due to limitations of the UDP service API, ICMP errors SHOULD be ignored. Packet Too Big errors [RFC4443] ("fragmentation needed and DF set" for IPv4 [RFC0792]) cannot properly occur and SHOULD be ignored if the implementation note in Section 4.6 is followed; otherwise, they SHOULD feed into a path MTU discovery algorithm [RFC4821]. Source Quench and Time Exceeded ICMP messages SHOULD be ignored. Host, network, port or protocol unreachable errors, or parameter problem errors MAY, after appropriate vetting, be used to inform the application of a failure in sending.

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 (including the case where the message is Empty, uses a code with a reserved class (1, 6 or 7), or has a message format error), it MUST reject the message; rejecting a Non-confirmable message MAY involve sending a matching Reset message, and apart from the Reset message the rejected message MUST be silently ignored.

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 multiple copies of a Non-confirmable message within

MAX_TRANSMIT_SPAN (limited by the provisions of Section 4.7, in particular by PROBING_RATE if no response is received), 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.)

Summarizing Section 4.2 and Section 4.3, the four message types can be used as in Table 1. "*" means that the combination is not used in normal operation, but only to elicit a Reset message ("CoAP ping").

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	*	-	X	X

Table 1: Usage of message types

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 (in communicating with the same endpoint) 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 (note that some receiving endpoints may not be able to distinguish unicast and multicast packets addressed to it, so endpoints generating Message IDs need to make sure these do not overlap). It is strongly recommended that the initial value of the variable (e.g., on startup) be randomized, in order to make successful off-path attacks on the protocol less likely.

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 might 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:

- A server might 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.

- A constrained server might 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 might 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 packet result in undesirable 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 needs to 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 [RFC4821]; 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 for messages not using DTLS security: 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, see Section 5.9.2.9) 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) MUST strictly limit the number of simultaneous outstanding interactions that they maintain to a given server (including proxies) to NSTART. 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 neither a response nor an Acknowledgment message has yet been received but is still expected (which may both occur at the same time, counting as one outstanding interaction). The default value of NSTART for this specification is 1.

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, and thus may allow a value for NSTART greater than one.

A client stops expecting a response to a Confirmable request for which no acknowledgment message was received, after EXCHANGE_LIFETIME. The specific algorithm by which a client stops to "expect" a response to a Confirmable request that was acknowledged, or to a Non-confirmable request, is not defined. Unless this is modified by additional congestion control optimizations, it MUST be chosen in such a way that an endpoint does not exceed an average data rate of PROBING_RATE in sending to another endpoint that does not respond.

Note: CoAP places the onus of congestion control mostly on the clients. However, clients may malfunction or actually be attackers, e.g. to perform amplification attacks (Section 11.3). To limit the damage (to the network and to its own energy resources), a server SHOULD implement some rate limiting for its response transmission based on reasonable assumptions about application requirements. This is most helpful if the rate limit can be made effective for the misbehaving endpoints, only.

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
NSTART	1

DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 Byte/second

Table 2: CoAP Protocol Parameters

4.8.1. Changing The Parameters

The values for ACK_TIMEOUT, ACK_RANDOM_FACTOR, MAX_RETRANSMIT, NSTART, DEFAULT_LEISURE (Section 8.2), and PROBING_RATE may be configured to values specific to the application environment (including dynamically adjusted values), 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 specific effects of operating with inconsistent values in an application environment are outside the scope of the present specification.

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 or increase NSTART, this can only be done safely when maintaining such measurements. Configurations MUST NOT decrease ACK_TIMEOUT or increase NSTART 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 than given here will require further adjustments in the time values (see Section 4.8.2).

If the choice of transmission parameters leads to an increase of derived time values (see Section 4.8.2), the configuration mechanism MUST ensure the adjusted value is also available to all the endpoints that these adjusted values are to be used to communicate with.

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:

- 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:

  ```
  ACK_TIMEOUT * ((2 ** MAX_RETRANSMIT) - 1) * ACK_RANDOM_FACTOR
  ```

- 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:

  ```
  ACK_TIMEOUT * ((2 ** (MAX_RETRANSMIT + 1)) - 1) *
  ACK_RANDOM_FACTOR
  ```

In addition, some assumptions need to be made on the characteristics of the network and the nodes.

- 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.

- 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.

- MAX_RTT is the maximum round-trip time, or:

  (2 * MAX_LATENCY) + PROCESSING_DELAY

From these values, we can derive the following values relevant to the protocol operation:

- 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:

  MAX_TRANSMIT_SPAN + (2 * MAX_LATENCY) + PROCESSING_DELAY

  or 247 seconds with the default transmission parameters.

- 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:

  MAX_TRANSMIT_SPAN + 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.

Table 3 summarizes the derived parameters introduced in this subsection with their default values.

name	default value
MAX_TRANSMIT_SPAN	45 s
MAX_TRANSMIT_WAIT	93 s
MAX_LATENCY	100 s
PROCESSING_DELAY	2 s
MAX_RTT	202 s
EXCHANGE_LIFETIME	247 s
NON_LIFETIME	145 s

Table 3: Derived Protocol Parameters

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 (Section 5.3, note that this is different from the Message ID that matches a Confirmable message to its Acknowledgement).

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).

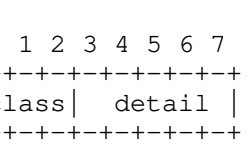


Figure 9: Structure of a Response Code

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).

As a human readable notation for specifications and protocol diagnostics, CoAP code numbers including the response code are documented in the format "c.dd", where "c" is the class in decimal, and "dd" is the detail as a two-digit decimal. For example, "Forbidden" is written as 4.03 -- indicating an 8-bit code value of hexadecimal 0x83 (4*0x20+3) or decimal 131 (4*32+3).

There are 3 classes of response codes:

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 are 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.

The possible response codes are described in detail in Section 5.9.

Responses can be sent in multiple ways, which are defined in the following subsections.

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, and no separate message is required to 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 (see also the discussion of PROCESSING_DELAY in Section 4.8.2). The Response to a request carried in a Non-confirmable message is always sent separately (as there is no Acknowledgement message).

One way to implement this in a server is to initiate the attempt to obtain the resource representation and, while that is in progress, time out an acknowledgement timer. A server may also immediately send an acknowledgement knowing in advance that there will be no piggy-backed response. In both cases, the acknowledgement effectively is a promise that the request will be acted upon later.

When the server finally has obtained the resource representation, it sends the response. When it is desired that this message is not lost, it is sent as a Confirmable message from the server to the client and answered by the client with an Acknowledgement, echoing the new Message ID chosen by the server. (It may also be sent as a Non-confirmable message; see Section 5.2.3.)

When the server chooses to use a separate response, it sends the Acknowledgement to the Confirmable request as an Empty message. Once the server sends back an Empty Acknowledgement, it MUST NOT send back the response in another Acknowledgement, even if the client retransmits another identical request. If a retransmitted request is received (perhaps because the original Acknowledgement was delayed), another Empty Acknowledgement is sent and any response MUST be sent as a separate response.

If the server then sends a Confirmable response, the client's Acknowledgement to that response MUST also be an Empty message (one that carries neither a request nor a response). The server MUST stop retransmitting its response on any matching Acknowledgement (silently ignoring any response code or payload) or Reset message.

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; for the purposes of terminating the retransmission sequence, this also serves as an acknowledgement. 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.)

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, along with additional address information of the corresponding endpoint.

5.3.1. Token

The Token is used to match a response with a request. The token value is a sequence of 0 to 8 bytes. (Note that every message carries a token, even if it is of zero length.) Every request carries a client-generated token, which the server MUST echo in any resulting response without modification.

A token is intended for use as a client-local identifier for differentiating between concurrent requests (see Section 5.3); it could have been called a "request ID".

The client SHOULD generate tokens in such a way that tokens currently in use for a given source/destination endpoint pair are unique. (Note that a client implementation can use the same token for any request if it uses a different endpoint each time, e.g. a different source port number.) 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 and receive piggy-backed responses. There are however multiple possible implementation strategies to fulfill this.

A client sending a request without using transport layer security (Section 9) SHOULD use a non-trivial, randomized token to guard against spoofing of responses (Section 11.4). This protective use of tokens is the reason they are allowed to be up to 8 bytes in size. The actual size of the random component to be used for the Token depends on the security requirements of the client and the level of threat posed by spoofing of responses. A client that is connected to the general Internet SHOULD use at least 32 bits of randomness; keeping in mind that not being directly connected to the Internet is not necessarily sufficient protection against spoofing. (Note that the Message ID adds little in protection as it is usually sequentially assigned, i.e. guessable, and can be circumvented by spoofing a separate response.) Clients that want to optimize the Token length may further want to detect the level of ongoing attacks (e.g., by tallying recent Token mismatches in incoming messages) and adjust the Token length upwards appropriately. [RFC4086] discusses randomness requirements for security.

An endpoint receiving a token it did not generate MUST treat it as opaque and make no assumptions about its content or structure.

5.3.2. Request/Response Matching Rules

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.

In case a message carrying a response is unexpected (the client is not waiting for a response from the identified endpoint, at the endpoint addressed, and/or with the given token), the response is rejected (Section 4.2, Section 4.3).

Implementation Note: A client that receives a response in a CON message may want to clean up the message state right after sending the ACK. If that ACK is lost and the server retransmits the CON, the client may no longer have any state to correlate this response to, making the retransmission an unexpected message; the client will likely send a Reset message so it does not receive any more retransmissions. This behavior is normal and not an indication of an error. (Clients that are not aggressively optimized in their state memory usage will still have message state that will identify the second CON as a retransmission. Clients that actually expect more messages from the server [I-D.ietf-core-observe] will have to keep state in any case.)

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:

- Content-Format
- ETag
- Location-Path
- Location-Query

- Max-Age
- Proxy-Uri
- Proxy-Scheme
- Uri-Host
- Uri-Path
- Uri-Port
- Uri-Query
- Accept
- If-Match
- If-None-Match
- Size1

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:

- Upon reception, unrecognized options of class "elective" MUST be silently ignored.
- 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 payload describing the unrecognized option(s) (see Section 5.5.2).

- Unrecognized options of class "critical" that occur in a Confirmable response, or piggy-backed in an Acknowledgement, MUST cause the response to be rejected (Section 4.2).

- Unrecognized options of class "critical" that occur in a Non-confirmable message MUST cause the message to be rejected (Section 4.3).

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 stop processing options they do not understand or implement.

Critical/Elective rules apply to non-proxying endpoints. A proxy processes options based on Unsafe/Safe-to-Forward classes as defined in Section 5.7.

5.4.2. Proxy Unsafe/Safe-to-Forward and NoCacheKey

In addition to an option being marked as Critical or Elective, options are also classified based on how a proxy is to deal with the option if it does not recognize it. For this purpose, an option can either be considered Unsafe to Forward (UnSafe is set) or Safe-to-Forward (UnSafe is clear).

In addition, for an option that is marked Safe-to-Forward, the option number indicates whether it is intended to be part of the Cache-Key (Section 5.6) in a request or not; if some of the NoCacheKey bits are 0, it is, if all NoCacheKey bits are 1, it is not (see Section 5.4.6).

Note: The Cache-Key indication is relevant only for proxies that do not implement the given option as a request option and instead rely on the Unsafe/Safe-to-Forward indication only. E.g., for ETag, actually using the request option as a part of the Cache-Key is grossly inefficient, but it is the best thing one can do if ETag is not implemented by a proxy, as the response is going to differ based on the presence of the request option. A more useful proxy that does implement the ETag request option is not using ETag as a part of the Cache-Key.

NoCacheKey is indicated in three bits so that only one out of eight codepoints is qualified as NoCacheKey, assuming this is the less likely case.

Proxy behavior with regard to these classes is defined in Section 5.7.

5.4.3. 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).

5.4.4. 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.

5.4.5. Repeatable Options

The definition of some options specifies that those options are 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, each supernumerary option occurrence that appears subsequently in the message MUST be treated like an unrecognized option (see Section 5.4.1).

5.4.6. Option Numbers

An Option is identified by an option number, which also provides some additional semantics information: e.g., 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.

More generally speaking, an Option number is constructed with a bit mask to indicate if an option is Critical/Elective, Unsafe/Safe-to-Forward and in the case of Safe-to-Forward, also a Cache-Key indication as shown by the following figure. In the following text, the bit mask is expressed as a single byte that is applied to the least significant byte of the option number in unsigned integer representation. When bit 7 (the least significant bit) is 1, an

option is Critical (and likewise Elective when 0). When bit 6 is 1, an option is Unsafe (and likewise Safe-to-Forward when 0). When bit 6 is 0, i.e., the option is not Unsafe, it is not a Cache-Key (NoCacheKey) if and only if bits 3-5 are all set to 1; all other bit combinations mean that it indeed is a Cache-Key. These classes of options are explained in the next sections.

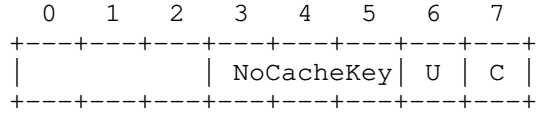


Figure 10: Option Number Mask (Least Significant Byte)

An endpoint may use an equivalent of the C code in Figure 11 to derive the characteristics of an option number "onum".

```
Critical = (onum & 1);
UnSafe = (onum & 2);
NoCacheKey = ((onum & 0x1e) == 0x1c);
```

Figure 11: Determining Characteristics from an Option Number

The option numbers for the options defined in this document are listed in the CoAP Option Number Registry (Section 12.2).

5.5. Payloads and Representations

Both requests and responses may include a 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 ("resource representation") or the result of the requested action ("action result"). Its format is specified by the Internet media type and content coding given by the Content-Format Option. In the absence of this option, no default value is assumed and the format will need to be inferred by the application (e.g., from the application context). Payload "sniffing" SHOULD only be attempted if no content type is given.

Implementation Note: On a quality of implementation level, there is a strong expectation that a Content-Format indication will be provided with resource representations whenever possible. This is not a "SHOULD"-level requirement solely because it is not a

protocol requirement, and it also would be difficult to outline exactly in what cases this expectation can be violated.

For responses indicating a client or server error, the payload is considered a representation of the result of the requested action only if a Content-Format Option is given. In the absence of this option, the payload is a Diagnostic Payload (Section 5.5.2).

5.5.2. Diagnostic Payload

If no Content-Format option is given, 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 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. In contrast to what is usual in HTTP, the payload SHOULD be empty if there is no additional information beyond the response code.

5.5.3. Selected Representation

Not all responses carry a payload that provides a representation of the resource addressed by the request. It is, however, sometimes useful to be able to refer to such a representation in relation to a response, independent of whether it actually was enclosed.

We use the term "selected representation" to refer to the current representation of a target resource that would have been selected in a successful response if the corresponding request had used the method GET and excluded any conditional request options (Section 5.10.8).

Certain response options provide metadata about the selected representation, which might differ from the representation included in the message for responses to some state-changing methods. Of the response options defined in this specification, only the ETag response option (Section 5.10.6) is defined as selected representation metadata.

5.5.4. Content Negotiation

A server may be able to supply a representation for a resource in one of multiple representation formats. Without further information from

the client, it will provide the representation in the format it prefers.

By using the Accept Option (Section 5.10.4) in a request, the client can indicate which content-format it prefers to receive.

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:

- the presented request method and that used to obtain the stored response match,
- 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 any request options marked as NoCacheKey (Section 5.4) or recognized by the Cache and fully interpreted with respect to its specified cache behavior (such as the ETag request option, Section 5.10.6, see also Section 5.4.2), and
- the stored response is either fresh or successfully validated as defined below.

The set of request options that is used for matching the cache entry is also collectively referred to as the "Cache-Key". For URI schemes other than coap and coaps, matching of those options that constitute the request URI may be performed under rules specific to the URI scheme.

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 5.10.5). 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.

If a client has a fresh stored response and makes a new request matching the request for that stored response, the new response invalidates the old response.

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 5.10.6) 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 it as described in 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

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.

In an overall architecture for a Constrained RESTful Environment, proxies can serve quite different purposes. Proxies can be explicitly selected by clients, a role that we term "forward-proxy". Proxies can also be inserted to stand in for origin servers, a role that we term "reverse-proxy". Orthogonal to this distinction, a proxy can map from a CoAP request to a CoAP request (CoAP-to-CoAP proxy) or translate from or to a different protocol ("cross-proxy"). Full definitions of these terms are provided in Section 1.2.

Notes: The terminology in this specification has been selected to be culturally compatible with the terminology used in the wider Web application environments, without necessarily matching it in every detail (which may not even be relevant to Constrained RESTful Environments). Not too much semantics should be ascribed to the components of the terms (such as "forward", "reverse", or "cross").

HTTP proxies, besides acting as HTTP proxies, often offer a transport protocol proxying function ("CONNECT") to enable end-to-end transport layer security through the proxy. No such function is defined for CoAP-to-CoAP proxies in this specification, as forwarding of UDP packets is unlikely to be of much value in Constrained RESTful environments. See also Section 10.2.7 for the cross-proxy case.

When a client uses a proxy to make a request that will use a secure URI scheme (e.g., coaps or https), the request towards the proxy SHOULD be sent using DTLS security except where equivalent lower layer security is used for the leg between the client and the proxy.

5.7.1. Proxy Operation

A proxy generally needs a way to determine potential request parameters for a request to a destination based on the request it received. This way is fully specified for a forward-proxy, but may depend on the specific configuration for a reverse-proxy. In particular, the client of a reverse-proxy generally does not indicate a locator for the destination, necessitating some form of namespace translation in the reverse-proxy. However, some aspects of the operation of proxies are common to all its forms.

If a 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. For options in the request that the proxy recognizes, it knows whether the option is intended to act as part of the key used in looking up the cached value or not. E.g., since requests for different Uri-Path values address different resources, Uri-Path values are always part of the Cache-Key, while, e.g., Token values are never part of the Cache-Key. For options that the proxy does not recognize but that are marked Safe-to-Forward in the option number, the option also indicates whether it is to be included in the Cache-Key (NoCacheKey is not all set) or not (NoCacheKey is all set). (Options that are unrecognized and marked Unsafe lead to 4.02 Bad Option.)

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 a response that cannot be processed by the proxy (e.g, due to unrecognized critical options, message format errors), 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, the generated (or implied) Max-Age Option MUST 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:

```
proxy-max-age = original-max-age - 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. Considering potential network delays on the way from the origin server, a proxy should be conservative in the max-age values offered.

All options present in a proxy request MUST be processed at the proxy. Unsafe 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. A CoAP-to-CoAP proxy MUST forward to the origin server all Safe-to-Forward options that it does not recognize. Similarly, Unsafe options in a response that are not recognized by the CoAP-to-CoAP proxy server MUST lead to a 5.02 (Bad Gateway) response. Again, Safe-to-Forward options that are not recognized MUST be forwarded.

Additional considerations for cross-protocol proxying between CoAP and HTTP are discussed in Section 10.

5.7.2. Forward-Proxies

CoAP distinguishes between requests made (as if) to an origin server and a request made through a forward-proxy. CoAP requests to a forward-proxy are made as normal Confirmable or Non-confirmable requests to the forward-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 5.10.2), 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 5.10.1); alternatively the URI in a proxy request can be assembled from a Proxy-Scheme option and the split options mentioned.

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 itself (see Section 5.10.2), then the request MUST be treated as a local (non-proxied) request.

Unless a proxy is configured to forward the proxy request to another proxy, it MUST translate the request as follows: The scheme of the request URI defines the outgoing protocol and its details (e.g., CoAP is used over UDP for the "coap" scheme and over DTLS for the "coaps" scheme.) For a CoAP-to-CoAP proxy, 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. This consumes the Proxy-Uri or Proxy-Scheme option, which is therefore not forwarded to the origin server.

5.7.3. Reverse-Proxies

Reverse-proxies do not make use of the Proxy-Uri or Proxy-Scheme options, but need to determine the destination (next hop) of a request from information in the request and information in their configuration. E.g., a reverse-proxy might offer various resources the existence of which it has learned through resource discovery as if they were its own resources. The reverse-proxy is free to build a namespace for the URIs that identify these resources. A reverse-proxy may also build a namespace that gives the client more control over where the request goes, e.g. by embedding host identifiers and port numbers into the URI path of the resources offered.

In processing the response, a reverse-proxy has to be careful that ETag option values from different sources are not mixed up on one resource offered to its clients. In many cases, the ETag can be forwarded unchanged. If the mapping from a resource offered by the

reverse-proxy to resources offered by its various origin servers is not unique, the reverse-proxy may need to generate a new ETag, making sure the semantics of this option are properly preserved.

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) piggy-backed 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 an Accept Option, that indicates the preferred content-format 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 code SHOULD be present in the response.

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 5.10.7). 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 and content coding given in the Content-Format Option, if provided.

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 code 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 code. 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 5.10.8.1) or If-None-Match (see Section 5.10.8.2) 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 code SHOULD be used 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 receiving this response 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 requests that cause the resource to cease being available, such as DELETE and in certain circumstances POST. 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 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, and MUST NOT include a payload.

When a cache that recognizes and processes the ETag response option receives a 2.03 (Valid) response, it MUST update the stored response with the value of the Max-Age Option included in the response (explicitly, or implicitly as a default value; see also Section 5.6.2). For each type of Safe-to-Forward option present in the response, the (possibly empty) set of options of this type that are present in the stored response MUST be replaced with the set of options of this type in the response received. (Unsafe options may trigger similar option specific processing as defined by the option.)

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 payload under the conditions 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 first 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 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".

The response SHOULD include a Size1 Option (Section 5.10.9) to indicate the maximum size of request entity the server is able and willing to handle, unless the server is not in a position to make this information available.

5.9.2.10. 4.15 Unsupported Content-Format

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 payload under the conditions 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 to indicate the number of seconds after which to retry.

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 forward-proxy for the URI specified in the Proxy-Uri Option or using Proxy-Scheme (see Section 5.10.2).

5.10. Option Definitions

The individual CoAP options are summarized in Table 4 and explained in the subsections of this section.

In this table, the C, U, and N columns indicate the properties, Critical, UnSafe, and NoCacheKey, respectively. Since NoCacheKey only has a meaning for options that are Safe-to-Forward (not marked Unsafe), the column is filled with a dash for UnSafe options. (The present specification does not define any NoCacheKey options, but the format of the table is intended to be useful for additional specifications.)

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical, U=Unsafe, N=NoCacheKey, R=Repeatable

Table 4: Options

5.10.1. 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:

- the Uri-Host Option specifies the Internet host of the resource being requested,
- the Uri-Port Option specifies the transport layer port number of the resource,
- each Uri-Path Option specifies one segment of the absolute path to the resource, and
- 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.

5.10.2. Proxy-Uri and Proxy-Scheme

The Proxy-Uri Option is used to make a request to a forward-proxy (see Section 5.7). The forward-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).

Note that the forward-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 forward-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 in a request containing the Proxy-Uri Option).

As a special case to simplify many proxy clients, the absolute-URI can be constructed from the Uri-* options. When a Proxy-Scheme Option is present, the absolute-URI is constructed as follows: A CoAP URI is constructed from the Uri-* options as defined in Section 6.5. In the resulting URI, the initial scheme up to, but not including the following colon is then replaced by the content of the Proxy-Scheme Option. Note that this case is only applicable if the components of the desired URI other than the scheme component actually can be expressed using Uri-* options; e.g., to represent a URI with a userinfo component in the authority, only Proxy-Uri can be used.

5.10.3. Content-Format

The Content-Format Option indicates the representation format of the message payload. The representation format is given as a numeric content format identifier that is defined in the CoAP Content Format Registry (Section 12.3). In the absence of the option, no default value is assumed, i.e. the representation format of any representation message payload is indeterminate (Section 5.5).

5.10.4. Accept

The CoAP Accept option can be used to indicate which Content-Format is acceptable to the client. The representation format is given as a numeric Content-Format identifier that is defined in the CoAP Content-Format Registry (Section 12.3). If no Accept option is 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 the Content-Format indicated. The server returns the preferred Content-Format if available. If the preferred Content-Format cannot be returned, then a 4.06 "Not Acceptable" MUST be sent as a response, unless another error code takes precedence for this response.

5.10.5. Max-Age

The Max-Age Option indicates the maximum time a response may be cached before it is 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.

The value is intended to be current at the time of transmission. Servers that provide resources with strict tolerances on the value of Max-Age SHOULD update the value before each retransmission. (See also Section 5.7.1.)

5.10.6. ETag

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 is generated by the server providing the resource, which may generate it 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 content or structure. (Endpoints that generate 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.)

5.10.6.1. ETag as a Response Option

The ETag Option in a response provides the current value (i.e., after the request was processed) of the entity-tag for the "tagged representation". If no Location-* options are present, the tagged representation is the selected representation (Section 5.5.3) of the target resource. If one or more Location-* options are present and

thus a location URI is indicated (Section 5.10.7), the tagged representation is the representation that would be retrieved by a GET request to the location URI.

An ETag response option can be included with any response for which there is a tagged representation (e.g., it would not be meaningful in a 4.04 or 4.00 response). The ETag Option MUST NOT occur more than once in a response.

There is no default value for the ETag Option; if it is not present in a response, the server makes no statement about the entity-tag for the tagged representation.

5.10.6.2. ETag as a Request Option

In a GET request, an endpoint that has one or more representations previously obtained from the resource, and has obtained ETag response options with these, can specify an instance of the ETag Option for one or more of these stored responses.

A server can issue a 2.03 Valid response (Section 5.9.1.3) in place of a 2.05 Content response if one of the ETags given is the entity-tag for the current representation, i.e. is valid; the 2.03 Valid response then echoes this specific ETag in a response option.

In effect, a client can determine if any of the stored representations is current (see Section 5.6.2) without needing to transfer them again.

The ETag Option MAY occur zero, one or more times in a request.

5.10.7. 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 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 that interprets these options and the implied URI identifies one or more currently stored responses, those entries MUST be marked as not fresh.

Each Location-Path Option specifies one segment of the absolute path to the resource, and each Location-Query 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, which is then interpreted relative to the request URI. Note that the relative URI-reference constructed this way always includes an absolute-path (e.g., leaving out Location-Path but supplying Location-Query means the path component in the URI is "/").

The options that are used to compute the relative URI-reference are collectively called Location-* options. Beyond Location-Path and Location-Query, more Location-* options may be defined in the future, and have been reserved option numbers 128, 132, 136, and 140. 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.

5.10.8. Conditional Request Options

Conditional request options enable a client to ask the server to perform the request only if certain conditions specified by the option are fulfilled.

For each of these options, if the condition given is not fulfilled, then the server MUST NOT perform the requested method. Instead, the server MUST respond with the 4.12 (Precondition Failed) response code.

If the condition is fulfilled, the server performs the request method as if the conditional request options were not present.

If the request would, without the conditional request options, result in anything other than a 2.xx or 4.12 response code, then any conditional request options MAY be ignored.

5.10.8.1. 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 If-Match option with an ETag matches a representation with that exact ETag. An If-Match option with an empty value matches any existing representation (i.e., it 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 options match, then the condition is fulfilled.

If there is one or more If-Match Option, but none of the options match, then the condition is not fulfilled.

5.10.8.2. 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 condition is not fulfilled.

(It is not very useful to combine If-Match and If-None-Match options in one request, because the condition will then never be fulfilled.)

5.10.9. Size1 Option

The Size1 option provides size information about the resource representation in a request. The option value is an integer number of bytes. Its main use is with block-wise transfers [I-D.ietf-core-block]. In the present specification, it is used in 4.13 responses (Section 5.9.2.9) to indicate the maximum size of request entity that the server is able and willing to handle.

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 component 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 in this section 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].

Implementation Note: Unfortunately, over time the URI format has acquired significant complexity. Implementers are encouraged to examine [RFC3986] closely. E.g., the ABNF for IPv6 addresses is more complicated than maybe expected. Also, implementers should take care to perform the processing of percent decoding/encoding exactly once on the way from a URI to its decoded components or back. Percent encoding is crucial for data transparency, but may lead to unusual results such as a slash in a path component.

6.1. coap URI Scheme

```
coap-URI = "coap:" "//" host [ ":" port ] path-abempty [ "?" query ]
```

If the host component 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; if a URI is received with a missing authority or an empty host, then it MUST be considered invalid. 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 through the use of DTLS as described in Section 9.1.

Considerations for caching of responses to "coaps" identified requests are discussed in Section 11.2.

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 bytes (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]. At this stage the URL is in ASCII encoding [RFC0020], even though the decoded components will be interpreted in UTF-8 [RFC3629] after step 5, 8 and 9.

   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 a Uri-Host Option is only used for a <host> component 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 each percent-encoding ("%" followed by two hexadecimal digits) to the corresponding byte.

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 each percent-encoding to the corresponding byte.

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. If the request does not include a Uri-Host Option, 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

As a part of discovering the services offered by a CoAP server, a client has to learn about the endpoint used by a server.

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 that offers resources 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 at other ports, e.g. in the dynamic port space.

Implementation Note: When a CoAP server is hosted by a 6LoWPAN node, header compression efficiency is improved when it also supports 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).

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. To maximize interoperability in a CoRE environment, a CoAP endpoint SHOULD support the CoRE Link Format of discoverable resources as described in [RFC6690], except where fully manual configuration is desired. 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 [RFC6690]. The Content-Format code "ct" attribute provides a hint about the Content-Formats this resource returns. Note that this is only a hint, and does not override the Content-Format Option of a CoAP response obtained by actually requesting the representation of the resource. 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-Format code attribute is present then nothing about the type can be assumed. The Content-Format code attribute MAY include a space-separated sequence of Content-Format codes, indicating that multiple content-formats are available. The syntax of the attribute value is summarized in the production ct-value in Figure 12, where cardinal, SP and DQUOTE are defined as in [RFC6690].

```
ct-value =  cardinal
         /  DQUOTE cardinal *( 1*SP cardinal ) DQUOTE
```

Figure 12

8. Multicast CoAP

CoAP supports making requests to a IP multicast group. This is defined by a series of deltas to Unicast CoAP. A more general discussion of group communication with CoAP is in [I-D.ietf-core-groupcomm].

CoAP endpoints that offer services that they want other endpoints to be able to find using multicast service discovery, join one or more of the appropriate all-CoAP-nodes multicast addresses (Section 12.8) and listen on the default CoAP port. Note that an endpoint might receive multicast requests on other multicast addresses, including the all-nodes IPv6 address (or via broadcast on IPv4); an endpoint MUST therefore be prepared to receive such messages but MAY ignore them if multicast service discovery is not desired.

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 endpoint. Such multicast requests MUST be Non-confirmable.

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.

To avoid an implosion of error responses, 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. Because such a Reset message will look identical to an RST for a unicast message from the sender, the sender MUST avoid using a Message ID that is also still active from this endpoint with any unicast endpoint that might receive the multicast message.

At the time of writing, multicast messages can only be carried in UDP, not in DTLS. This means that the security modes defined for CoAP in this document are not applicable to multicast.

8.2. Request/Response Layer

When a server is aware that a request arrived via multicast, the server MAY always ignore 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 [RFC6690] query filtering, a server should not respond to a multicast request if the filter does not match. More examples are in [I-D.ietf-core-groupcomm].)

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. If further responses need to be sent based on the same multicast address membership, a new leisure period starts at the earliest after the previous one finishes.

To compute a value for Leisure, the server should have a group size estimate G, a target data transfer 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 8 kbit/s = 1 kB/s. The resulting lower bound for the Leisure is 10 seconds.

If a CoAP endpoint does not have suitable data to compute a value for Leisure, it MAY resort to DEFAULT_LEISURE.

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.

For the purposes of interpreting the Location-* options and any links embedded in the representation and, the request URI (base URI) relative to which the response is interpreted, is formed by replacing the multicast address in the Host component of the original request URI by the literal IP address of the endpoint actually responding.

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 a 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 or URI constructed from Proxy-Scheme 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.

This specification does not provide a way to indicate the unicast-modified request URI (base URI) in responses thus forwarded. Proxying multicast requests is discussed in more detail in [I-D.ietf-core-groupcomm]; one proposal to address the base URI issue can be found in section 3 of [I-D.bormann-coap-misc].

9. Securing CoAP

This section defines the DTLS binding for CoAP.

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 [I-D.bormann-core-ipsec-for-coap]. Certain link layers in use with constrained nodes also provide link layer security, which may be appropriate with proper key management.

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). Conversely, if more than two entities share a specific pre-shared key, this key only enables the entities to authenticate as a member of that group and not as a specific peer.

RawPublicKey: DTLS is enabled and the device has an asymmetric key pair without a certificate (a raw public key) 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 13). 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.

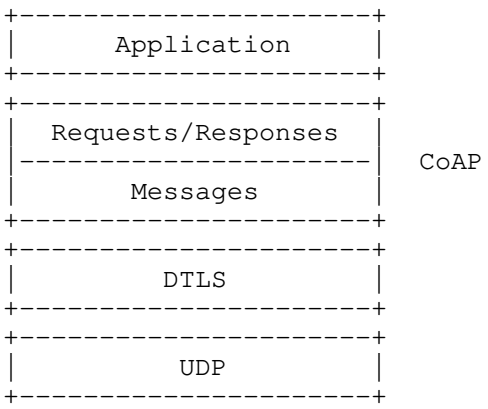


Figure 13: Abstract layering of DTLS-secured CoAP

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, all modes of DTLS may not be applicable. Some 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 [RFC6655]), integrity check values (e.g., 8 bytes with TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]) 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. (For some modes of using DTLS, this specification identifies a mandatory to implement cipher suite. This is an implementation requirement to maximize interoperability in those cases where these cipher suites are indeed appropriate. The specific security policies of an application may determine the actual (set of) cipher suites that can be used.) 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. 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: 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.

9.1.2. 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.

This means the response to a DTLS secured request MUST always be DTLS secured using the same security session and epoch. Any attempt to supply a NoSec response to a DTLS request simply does not match the

request and (unless it does match an unrelated NoSec request) therefore MUST be rejected.

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 [RFC6655].

Depending on the commissioning model, applications may need to define an application profile for identity hints as required and detailed in [RFC4279] (Section 5.2) to enable the use of PSK identity hints.

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); e.g., the asymmetric key pair is generated by the manufacturer and installed on the device (see also Section 11.6). 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.mcgrew-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. The key used MUST be ECDSA-capable. The curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. Implementations MUST use the Supported Elliptic Curves Extension and Supported Point Format extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090]

can be used as an implementation method. Some guidance relevant to the implementation of this cipher suite can be found in [W3CXMLSEC]. The mechanism for using raw public keys with TLS is specified in [I-D.ietf-tls-oob-pubkey].

Implementation Note: Specifically, this means the extensions listed in Figure 14 with at least the values listed will be present in the DTLS handshake.

```
Extension: elliptic_curves
 Type: elliptic_curves (0x000a)
 Length: 4
 Elliptic Curves Length: 2
 Elliptic curves (1 curve)
   Elliptic curve: secp256r1 (0x0017)

Extension: ec_point_formats
 Type: ec_point_formats (0x000b)
 Length: 2
 EC point formats Length: 1
 Elliptic curves point formats (1)
   EC point format: uncompressed (0)

Extension: signature_algorithms
 Type: signature_algorithms (0x000d)
 Length: 4
 Data (4 bytes): 00 02 04 03
   HashAlgorithm: sha256 (4)
   SignatureAlgorithm: ecdsa (3)
```

Figure 14: DTLS extensions present for TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

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 by the endpoint from the public key as described in Section 2 of [RFC6920]. 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

[RFC6920] 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 (initial and ongoing) provisioning, an access control list of identifiers the device may start DTLS sessions with SHOULD also be installed and maintained.

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 [I-D.mcgrew-tls-aes-ccm-ecc], [RFC5246], [RFC4492]. Namely, the certificate includes a SubjectPublicKeyInfo that indicates an algorithm of id-ecPublicKey with namedCurves secp256r1 [RFC5480]; the public key format is uncompressed [RFC5480]; the hash algorithm is SHA-256; if included the key usage extension indicates digitalSignature. Certificates MUST be signed with ECDSA using secp256r1, and the signature MUST use SHA-256. The key used MUST be ECDSA-capable. The curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. Implementations MUST use the Supported Elliptic Curves Extension and Supported Point Format extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090] can be used as an implementation method.

The Authority Name in the certificate would be built out of a long term unique identifier for the device such as the EUI-64 [EUI64]. The Authority Name could also be based on the FQDN that was used as the Host part of the CoAP URI. However, the device's IP address should not typically be used as the Authority name as it would change over time. 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 that the validity dates of the certificate are within range. The certificate MUST be validated as appropriate for the security requirements, using functionality equivalent to the algorithm specified in [RFC5280] section 6. 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 field of URI type 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.

CoRE support for certificate status checking requires further study. As a mapping of OCSP [RFC2560] onto CoAP is not currently defined and OCSP may also not be easily applicable in all environments, an alternative approach may be using the TLS Certificate Status Request extension ([RFC6066] section 8, also known as "OCSP stapling") or preferably the Multiple Certificate Status Extension ([I-D.ietf-tls-multiple-cert-status-extension]), if available.

If the system has a shared key in addition to the certificate, then a cipher suite that includes the shared key such as TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA [RFC5489] SHOULD be used.

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 or Proxy-Scheme 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. However, similar considerations apply to reverse-proxies as to forward-proxies, and there generally will be an expectation that reverse-proxies operate in a similar way forward-proxies would. As an implementation note, HTTP client libraries may make it hard to operate an HTTP-CoAP forward proxy by not providing a way to put a CoAP URI on the HTTP Request-Line; reverse-proxying may therefore lead to wider applicability of a proxy. A separate specification may define a convention for URIs operating such a HTTP-CoAP reverse proxy [I-D.castellani-core-http-mapping].

10.1. CoAP-HTTP Proxying

If a request contains a Proxy-Uri or Proxy-Scheme 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. (See also Section 5.7 for how the request to the proxy is formulated, including security requirements.)

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 in the subsections of this section.

If the proxy is unable or unwilling to service a request with an HTTP URI, a 5.05 (Proxying Not Supported) response is 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 is returned; if a result can be obtained but is not understood, a 5.02 (Bad Gateway) response is 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 code SHOULD be returned. The payload of the response MUST be a representation of the target HTTP resource, and the Content-Format 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 an Accept Option, identifying the preferred response content-format.

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. Note that CoAP ETags are always strong ETags in the HTTP sense; CoAP does not have the equivalent of HTTP weak ETags, and there is no good way to make use of these in a cross-proxy.

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.

10.1.3. 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.

10.1.4. 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 Proxying

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. Unless otherwise specified all the statements made are RECOMMENDED behavior; some highly constrained implementations may need to resort to shortcuts. 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 in the subsections of this section.

If the proxy is unable or unwilling to service a request with a CoAP URI, a 501 (Not Implemented) response is 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 is returned; if a result can be obtained but is not understood, a 502 (Bad Gateway) response is 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 is returned. The payload of the response MUST be a representation of the target CoAP resource, and the Content-Type and Content-Encoding header fields be set accordingly. The response MUST indicate a max-age directive that indicates a value no greater than the remaining time the representation can be considered fresh. If the CoAP response has an ETag option, the proxy should include an ETag header field in the response.

A client can influence the processing of a GET request by including the following options:

Accept: The most preferred Media-type of the HTTP Accept header field 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 sends a 406 (not acceptable) response. The proxy MAY then retry the request with further Media-types from the HTTP Accept header field.

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 are 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.

Implementation Note: An HTTP-CoAP proxy may want to try using a block-wise transfer [I-D.ietf-core-block] option to minimize the amount of data actually transferred, but needs to be prepared for the case that the origin server does not support block-wise transfers.

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.

If any of the Location-* Options are present in the CoAP response, a Location header field constructed from the values of these options is 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 is returned to the client. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes is 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 is 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 yet been specified. For now, a 501 (Not Implemented) error is 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. CoAP access control implementations need to ensure they don't introduce vulnerabilities through discrepancies between the code deriving access control decisions from a URI and the code finally serving up the resource addressed by the URI. The most complex parser remaining could be the one for the CoRE Link Format, although this also has been designed with a goal of reduced implementation complexity [RFC6690]. (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.

For a caching implementation, any access control considerations that would apply to making the request that generated the cache entry also need to be applied to the value in the cache. This is relevant for clients that implement multiple security domains, as well as for proxies that may serve multiple clients. Also, a caching proxy MUST NOT make cached values available to requests that have lesser transport security properties than to which it would make available the process of forwarding the request in the first place.

Unlike the "coap" scheme, responses to "coaps" identified requests are never "public" and thus MUST NOT be reused for shared caching unless the cache is able to make equivalent access control decisions to the ones that led to the cached entry. They can, however, be reused in a private cache if the message is cacheable by default in CoAP.

Finally, a proxy that fans out Separate Responses (as opposed to Piggy-backed Responses) to multiple original requesters may provide additional amplification (see Section 11.3).

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.

Therefore, large amplification factors SHOULD NOT be provided in the response if the request is not authenticated. 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 in some way, cryptographically or by some multicast boundary limiting the potential sources. If possible a CoAP server SHOULD limit the support for multicast requests to the 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 an ACK in response to a CON message, thus potentially preventing the sender of the CON message from retransmitting, and drowning out the actual response; or

3. 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

4. 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

5. spoofing observe messages, etc.

Response spoofing by off-path attackers can be detected and mitigated even without transport layer security by choosing a non-trivial, randomized token in the request (Section 5.3.1). [RFC4086] discusses randomness requirements for security.

In principle, other kinds of 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.

With or without source address spoofing, a client can attempt to overload a server by sending requests, preferably complex ones, to a server; address spoofing makes tracing back, and blocking, this attack harder. Given that the cost of a CON request is small, this attack can easily be executed. Under this attack, a constrained node with limited total energy available may exhaust that energy much more quickly than planned (battery depletion attack). Also, if the client uses a Confirmable message and the server responds with a Confirmable separate response to a (possibly spoofed) address that does not respond, the server will have to allocate buffer and retransmission logic for each response up to the exhaustion of MAX_TRANSMIT_SPAN, making it more likely that it runs out of resources for processing legitimate traffic. The latter problem can be mitigated somewhat by limiting the rate of responses as discussed in Section 4.7. An attacker could also spoof the address of a legitimate client, which, if the server uses separate responses, might block legitimate responses to that client because of NSTART=1. All these attacks can be prevented using a security mode other than NoSec, leaving only attacks on the security protocol.

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 against a victim listening to UDP packets at a given address (IP address and port):

- the attacker sends a message to a CoAP endpoint with the given address as the fake source address,
- the CoAP endpoint replies with a message to the given source address,
- the victim at the given 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, or possibly as a Token. 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.

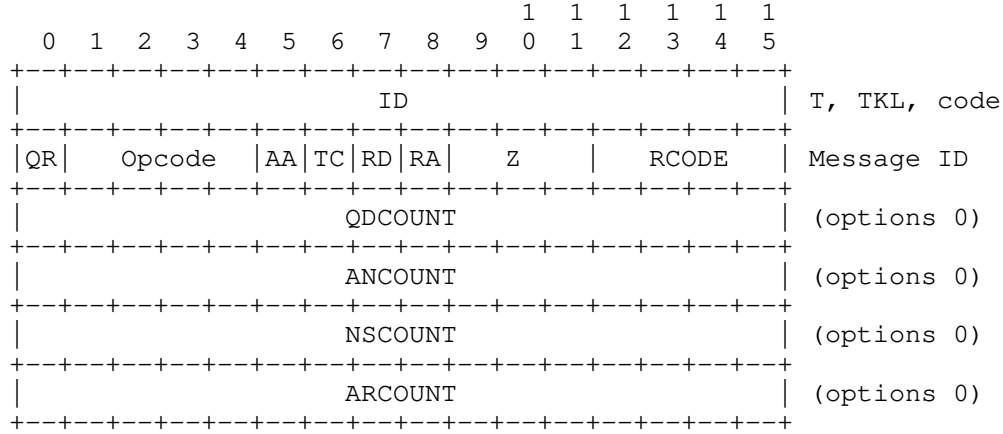


Figure 15: DNS Header vs. CoAP Message

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.

11.6. Constrained node considerations

Implementers on constrained nodes often find themselves without a good source of entropy [RFC4086]. If that is the case, the node MUST NOT be used for processes that require good entropy, such as key generation. Instead, keys should be generated externally and added to the device during manufacturing or commissioning.

Due to their low processing power, constrained nodes are particularly susceptible to timing attacks. Special care must be taken in implementation of cryptographic primitives.

Large numbers of constrained nodes will be installed in exposed environments and will have little resistance to tampering, including recovery of keying materials. This needs to be considered when defining the scope of credentials assigned to them. In particular, assigning a shared key to a group of nodes may make any single constrained node a target for subverting the entire group.

12. IANA Considerations

12.1. CoAP Code Registries

This document defines two sub-registries for the values of the Code field in the CoAP header within the Constrained RESTful Environments (CoRE) Parameters ("CoRE Parameters") registry.

Values in the two sub-registries are eight-bit values notated as three decimal digits c.dd separated by a period between the first and the second digit; the first digit c is between 0 and 7 and denotes the code class; the second and third digit dd denote a decimal number between 00 and 31 for the detail.

All Code values are assigned by sub-registries according to the following ranges:

0.00 Indicates an Empty message (see Section 4.1).

0.01-0.31 Indicates a request. Values in this range are assigned by the "CoAP Method Codes" sub-registry (see Section 12.1.1).

1.00-1.31 Reserved

2.00-5.31 Indicates a response. Values in this range are assigned by the "CoAP Response Codes" sub-registry (see Section 12.1.2).

6.00-7.31 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 0.01-0.31, 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
0.01	GET	[RFCXXXX]
0.02	POST	[RFCXXXX]
0.03	PUT	[RFCXXXX]
0.04	DELETE	[RFCXXXX]

Table 5: CoAP Method Codes

All other Method Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a method code should specify the semantics of a request with that code, including the following properties:

- The response codes the method returns in the success case.
- 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 2.00-5.31, 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
2.01	Created	[RFCXXXX]
2.02	Deleted	[RFCXXXX]
2.03	Valid	[RFCXXXX]
2.04	Changed	[RFCXXXX]
2.05	Content	[RFCXXXX]
4.00	Bad Request	[RFCXXXX]
4.01	Unauthorized	[RFCXXXX]
4.02	Bad Option	[RFCXXXX]
4.03	Forbidden	[RFCXXXX]
4.04	Not Found	[RFCXXXX]
4.05	Method Not Allowed	[RFCXXXX]
4.06	Not Acceptable	[RFCXXXX]
4.12	Precondition Failed	[RFCXXXX]
4.13	Request Entity Too Large	[RFCXXXX]
4.15	Unsupported Content-Format	[RFCXXXX]
5.00	Internal Server Error	[RFCXXXX]
5.01	Not Implemented	[RFCXXXX]
5.02	Bad Gateway	[RFCXXXX]
5.03	Service Unavailable	[RFCXXXX]
5.04	Gateway Timeout	[RFCXXXX]
5.05	Proxying Not Supported	[RFCXXXX]

Table 6: CoAP Response Codes

The Response Codes 3.00-3.31 are Reserved for future use. All other Response Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG approval" as described in [RFC5226].

The documentation of a response code should specify the semantics of a response with that code, including the following properties:

- The methods the response code applies to.

- Whether payload is required, optional or not allowed.

- 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 payload.

- The format of the payload. For example, the format in a 2.05 (Content) response is indicated by the Content-Format Option; the format of the payload in an error response is always Net-Unicode text.

- Whether the response is cacheable according to the freshness model.

- Whether the response is validatable according to the validation model.

- 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 sub-registry for the Option Numbers used in CoAP options within the "CoRE Parameters" registry. The name of the sub-registry is "CoAP Option Numbers".

Each entry in the sub-registry must include the Option Number, the name of the option and a reference to the option's documentation.

Initial entries in this sub-registry are as follows:

Number	Name	Reference
0	(Reserved)	[RFCXXXX]
1	If-Match	[RFCXXXX]
3	Uri-Host	[RFCXXXX]
4	ETag	[RFCXXXX]
5	If-None-Match	[RFCXXXX]
7	Uri-Port	[RFCXXXX]
8	Location-Path	[RFCXXXX]
11	Uri-Path	[RFCXXXX]
12	Content-Format	[RFCXXXX]
14	Max-Age	[RFCXXXX]
15	Uri-Query	[RFCXXXX]
17	Accept	[RFCXXXX]
20	Location-Query	[RFCXXXX]
35	Proxy-Uri	[RFCXXXX]
39	Proxy-Scheme	[RFCXXXX]

	60	Size1	[RFCXXXX]
	128	(Reserved)	[RFCXXXX]
	132	(Reserved)	[RFCXXXX]
	136	(Reserved)	[RFCXXXX]
	140	(Reserved)	[RFCXXXX]

Table 7: CoAP Option Numbers

The IANA policy for future additions to this sub-registry is split into three tiers as follows. The range of 0..255 is reserved for options defined by the IETF (IETF Review or IESG approval). The range of 256..2047 is reserved for commonly used options with public specifications (Specification Required). The range of 2048..64999 is for all other options including private or vendor specific ones, which undergo a Designated Expert review to help ensure that the option semantics are defined correctly. The option numbers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments.

Option Number	Policy [RFC5226]
0..255	IETF Review or IESG approval
256..2047	Specification Required
2048..64999	Designated Expert
65000..65535	Reserved for experiments

Table 8: CoAP Option Number Registry Policy

The documentation of an Option Number should specify the semantics of an option with that number, including the following properties:

- The meaning of the option in a request.
- The meaning of the option in a response.
- Whether the option is critical or elective, as determined by the Option Number.
- Whether the option is Safe-to-Forward, and, if yes, whether it is part of the Cache-Key, as determined by the Option Number (see Section 5.4.2).
- The format and length of the option's value.

- Whether the option must occur at most once or whether it can occur multiple times.

- 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 5.4.4).

12.3. Content-Format Registry

Internet 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 sub-registry for a subset of Internet media types to be used in CoAP and assigns each, in combination with a content-coding, a numeric identifier. The name of the sub-registry is "CoAP Content-Formats", within the "CoRE Parameters" registry.

Each entry in the sub-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-coding 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 separate 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 Content-Format identifier for each is to be registered. Similarly, other parameters related to an Internet media type, such as level, can be defined for a CoAP Content-Format entry.

Initial entries in this sub-registry are as follows:

Media type	Encoding	Id.	Reference
text/plain; charset=utf-8	-	0	[RFC2046][RFC3676][RFC5147]
application/link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/octet-stream	-	42	[RFC2045][RFC2046]
application/exi	-	47	[EXIMIME]
application/json	-	50	[RFC4627]

Table 9: CoAP Content-Formats

The identifiers between 65000 and 65535 inclusive are reserved for experiments. They are not meant for vendor specific use of any kind and MUST NOT be used in operational deployments. The identifiers between 256 and 9999 are reserved for future use in IETF specifications (IETF review or IESG approval). 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 0-255 inclusive to the sub-registry is "Expert Review" as described in [RFC5226]. The IANA policy for additions in the range 10000-64999 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; CoAP needs the Link-Local and Site-Local scopes only. The address should be of the form FF0x::nn, where nn is a single byte, to ensure good compression of the local-scope address with [RFC6282].

[The explanatory text to be removed upon allocation of the addresses, except for the note about the distinct multicast addresses.]

13. Acknowledgements

Brian Frank was a contributor to and co-author of previous drafts of this specification.

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 Beroset, Angelo P. Castellani, Gilbert Clark, Robert Cragie, Esko Dijk, Lisa Dusseault, Mehmet Ersue, 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, Dan Romascanu, 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.

Special thanks also to the IESG reviewers, Adrian Farrel, Martin Stiemerling, Pete Resnick, Richard Barnes, Sean Turner, Spencer Dawkins, Stephen Farrell, and Ted Lemon, who contributed in-depth reviews.

Some of the text has been borrowed from the working documents of the IETF httpbis working group.

14. References

14.1. Normative References

[I-D.ietf-tls-oob-pubkey]
Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Out-of-Band Public Key Validation for Transport Layer Security (TLS)", draft-ietf-tls-oob-pubkey-07 (work in progress), February 2013.

[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-06 (work in progress), February 2013.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC2045] Freed, N. and N.S. 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.

[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.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[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.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[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.

[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.

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

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, April 2013.

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>.

[HHGTTG] Adams, D., "The Hitchhiker's Guide to the Galaxy", October 1979.

[I-D.allman-tcpm-rto-consider] Allman, M., "Retransmission Timeout Considerations", draft-allman-tcpm-rto-consider-01 (work in progress), May 2012.

[I-D.bormann-coap-misc] Bormann, C. and K. Hartke, "Miscellaneous additions to CoAP", draft-bormann-coap-misc-22 (work in progress), December 2012.

[I-D.bormann-core-ipsec-for-coap] Bormann, C., "Using CoAP with IPsec", draft-bormann-core-ipsec-for-coap-00 (work in progress), December 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.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-groupcomm]
Rahman, A. and E. Dijk, "Group Communication for CoAP", draft-ietf-core-groupcomm-06 (work in progress), April 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-lwig-terminology]
Bormann, C., Ersue, M., and A. Keraenen, "Terminology for Constrained Node Networks", draft-ietf-lwig-terminology-04 (work in progress), April 2013.

[I-D.ietf-tls-multiple-cert-status-extension]
Pettersen, Y., "The TLS Multiple Certificate Status Request Extension", draft-ietf-tls-multiple-cert-status-extension-08 (work in progress), April 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>.

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[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.

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[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.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 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.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, March 2009.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 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.

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July 2012.

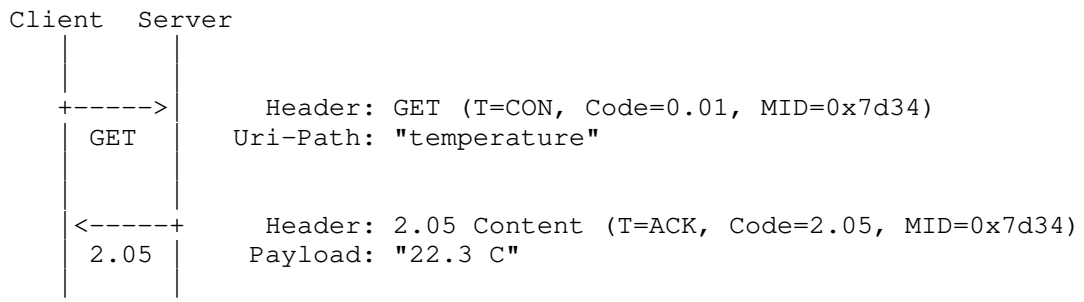
[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, April 2013.

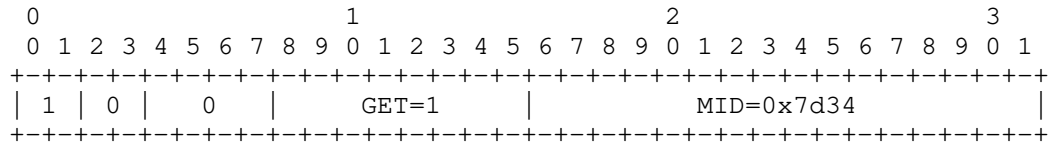
[W3CXMLSEC] Wenning, R., "Report of the XML Security PAG", October 2012, <http://www.w3.org/2011/xmlsec-pag/pagreport.html>.

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 16 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 + 11 = 11, Length 11, Value "temperature"); the Token is left 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 empty Token value. The response includes a Payload of "22.3 C" and is 11 bytes long.





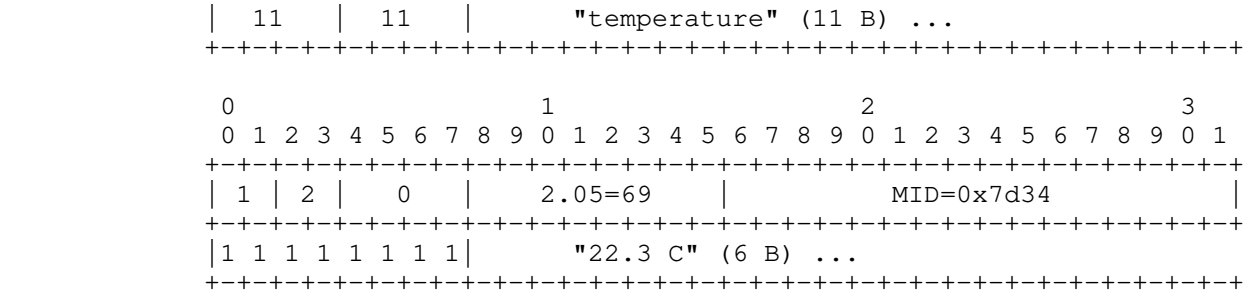
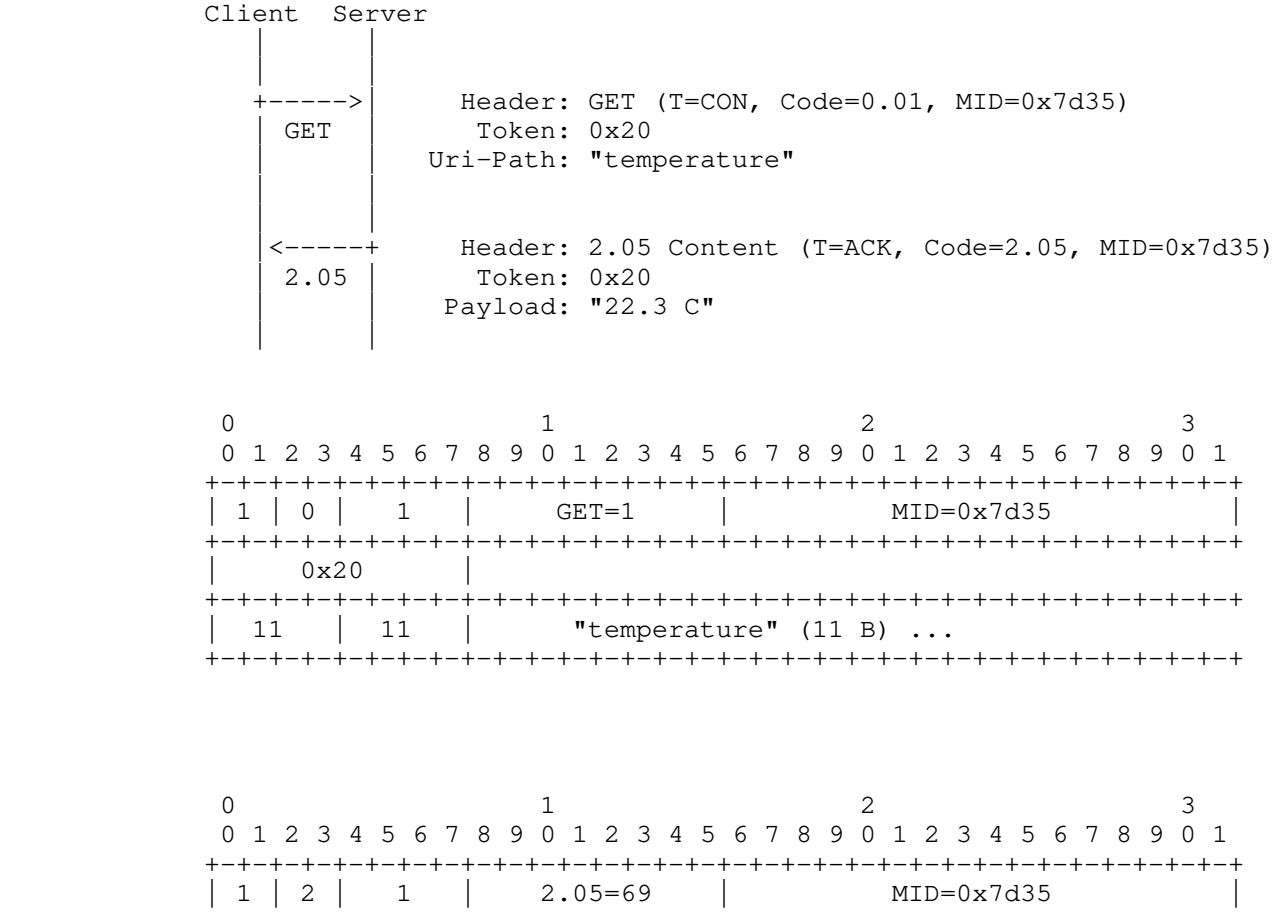


Figure 16: Confirmable request; piggy-backed response

Figure 17 shows a similar example, but with the inclusion of an non-empty Token (Value 0x20) in the request and the response, increasing the sizes to 17 and 12 bytes, respectively.



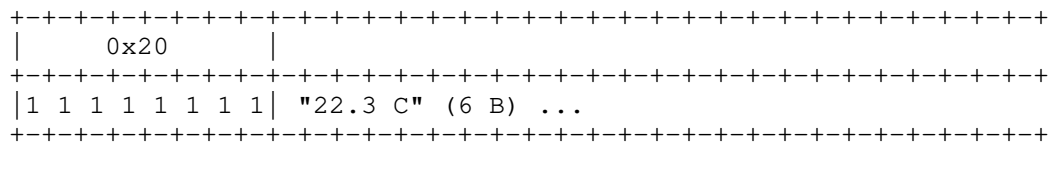


Figure 17: Confirmable request; piggy-backed response

In Figure 18, 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.

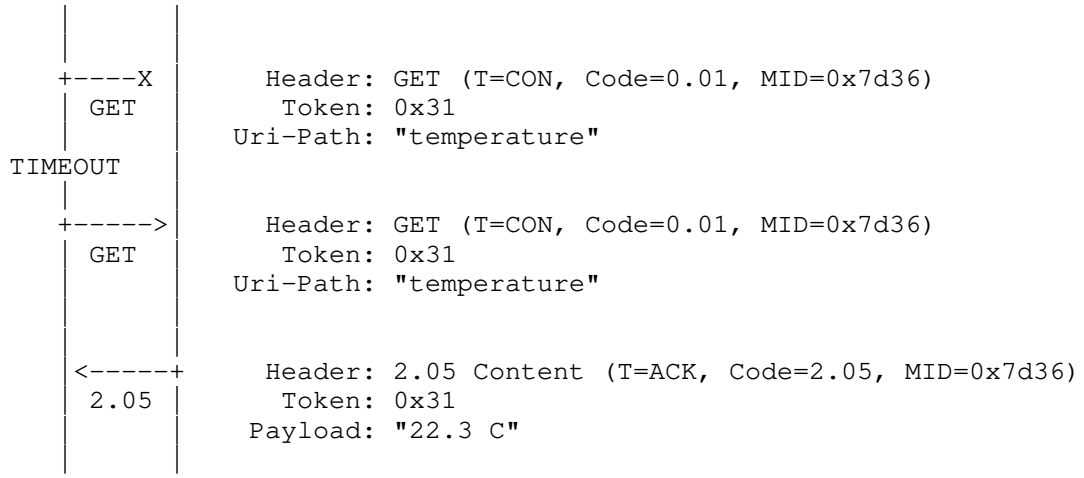
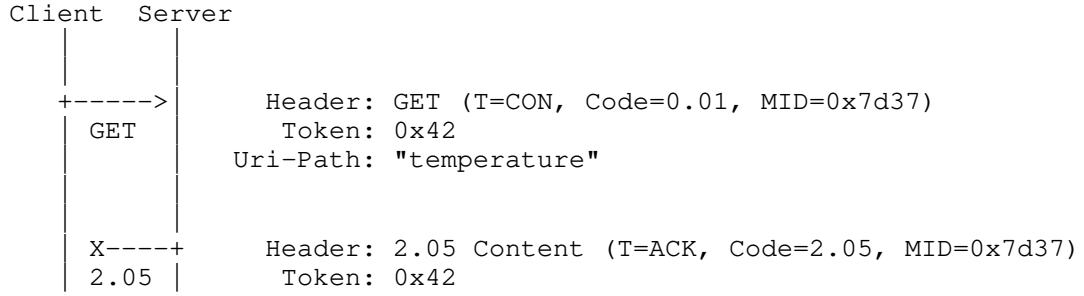


Figure 18: Confirmable request (retransmitted); piggy-backed response

In Figure 19, the first Acknowledgement message from the server to the client is lost. After ACK_TIMEOUT seconds, the client retransmits the request.



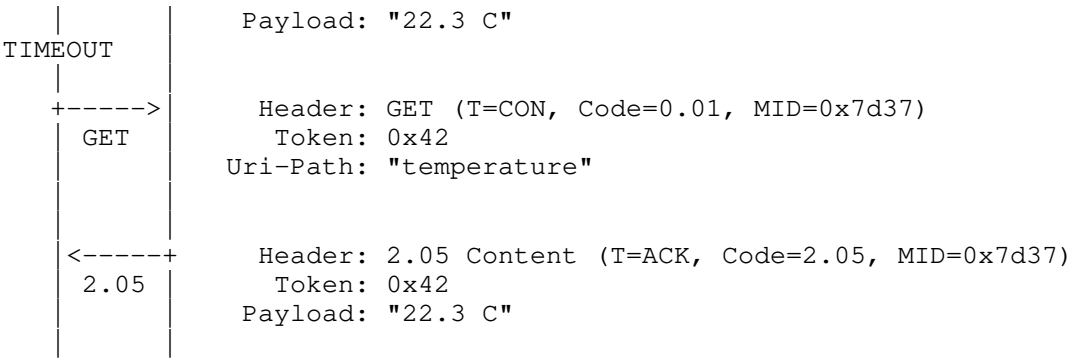


Figure 19: Confirmable request; piggy-backed response (retransmitted)

In Figure 20, 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.

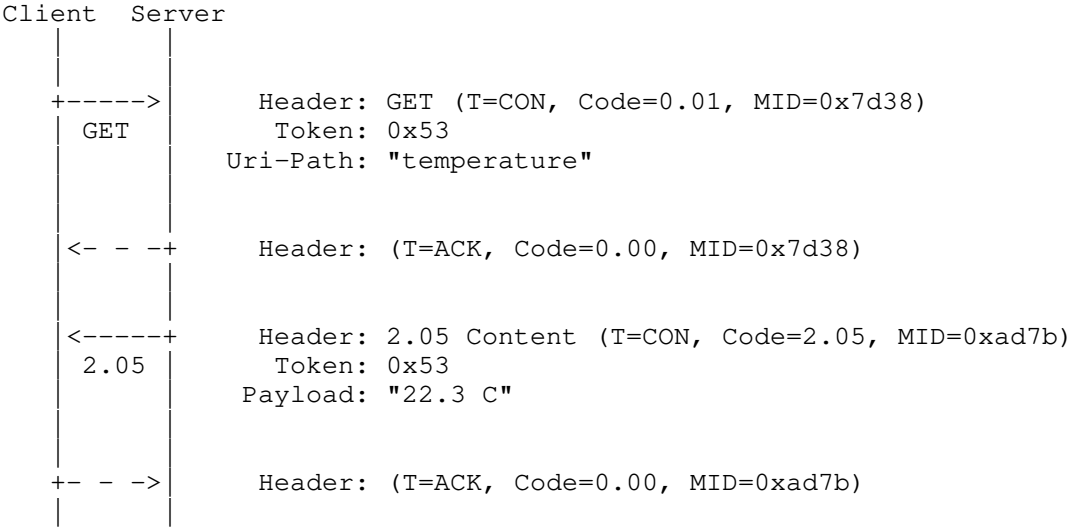


Figure 20: Confirmable request; separate response

Figure 21 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.

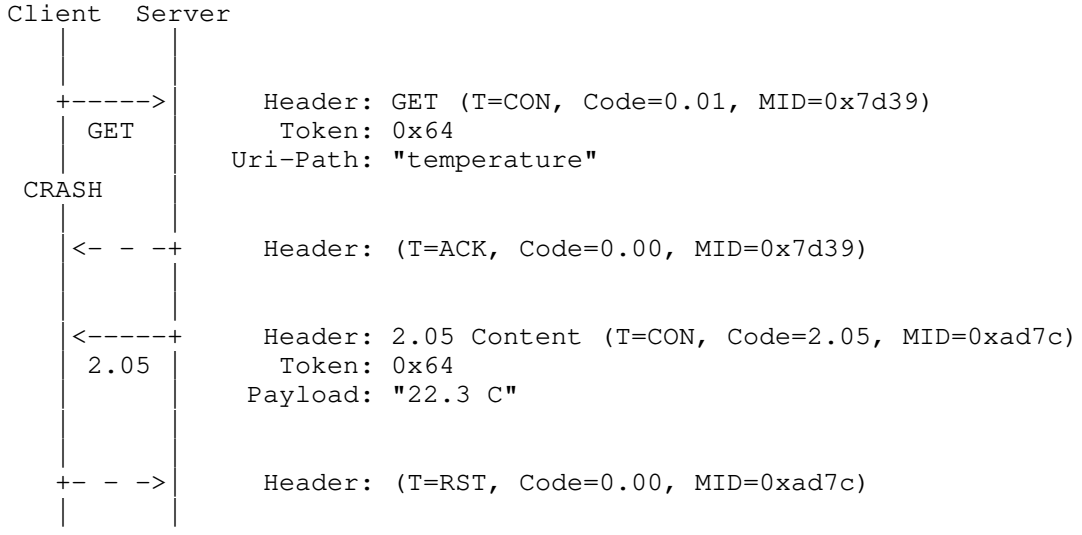


Figure 21: Confirmable request; separate response (unexpected)

Figure 22 shows a basic GET request where the request and the response are Non-confirmable, so both may be lost without notice.

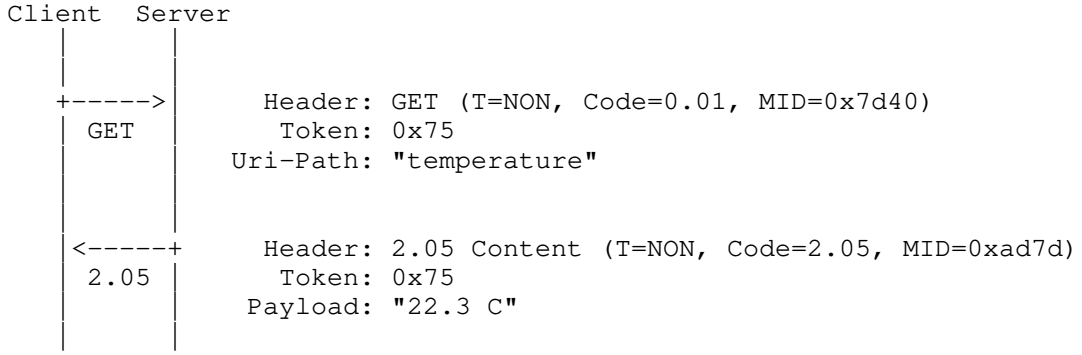


Figure 22: Non-confirmable request; Non-confirmable response

In Figure 23, 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.

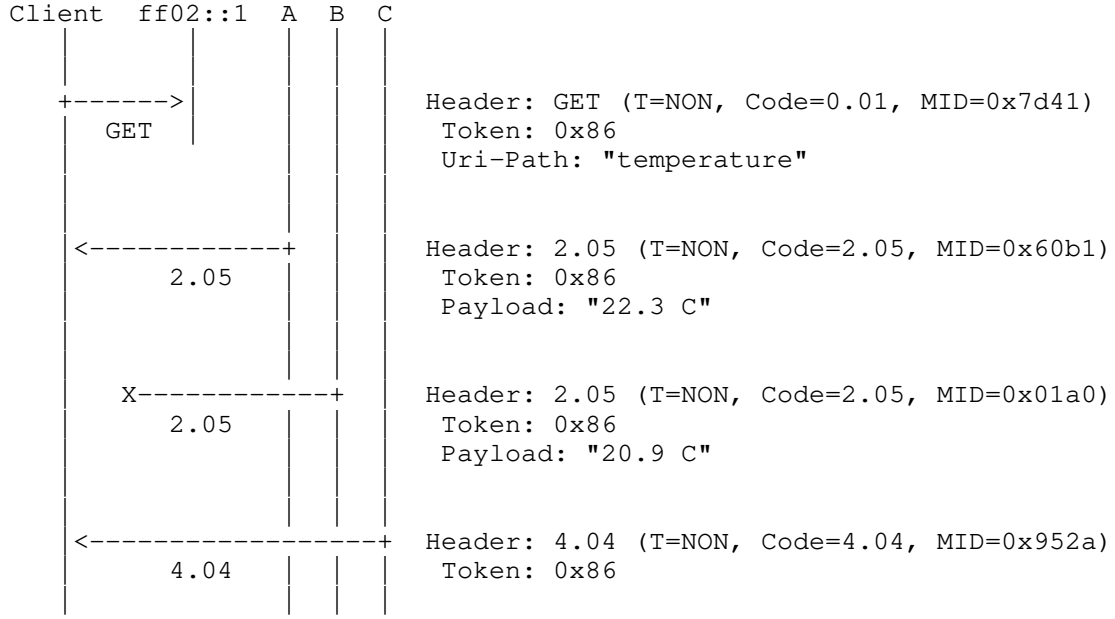


Figure 23: Non-confirmable request (multicast); Non-confirmable response

Appendix B. URI Examples

The following examples demonstrate different sets of Uri options, and the result after constructing an URI from them. In addition to the options, Section 6.5 refers to the destination IP address and port, but not all paths of the algorithm cause the destination IP address and port to be included in the URI.

- Input:

  Destination IP Address = [2001:db8::2:1]
  Destination UDP Port = 5683

  Output:

  coap://[2001:db8::2:1]/

- Input:

  Destination IP Address = [2001:db8::2:1]
  Destination UDP Port = 5683
  Uri-Host = "example.net"

Output:

coap://example.net/

- Input:

  Destination IP Address = [2001:db8::2:1]
  Destination UDP Port = 5683
  Uri-Host = "example.net"
  Uri-Path = ".well-known"
  Uri-Path = "core"

  Output:

  coap://example.net/.well-known/core

- Input:

  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

  Output:

  coap://xn--18j4d.example/
  %E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF

  (The line break has been inserted for readability; it is not part of the URI.)

- Input:

  Destination IP Address = 198.51.100.1
  Destination UDP Port = 61616
  Uri-Path = ""
  Uri-Path = "/"
  Uri-Path = ""
  Uri-Path = ""
  Uri-Query = "//"
  Uri-Query = "?&"

  Output:

  coap://198.51.100.1:61616//%2F//?%2F%2F&?%26

Appendix C. Changelog

(To be removed by RFC editor before publication.)

Changes from ietf-17 to ietf-18: Address comments from the IESG reviews.

- Accept is now critical.
- Add Size1 option for 4.13 responses.

Changes from ietf-15 to ietf-16: Address comments from the IESG reviews. These should not impact interoperability.

- Clarify that once there has been an empty ACK, all further ACKs to the same message also must be empty (#301).
- Define Cache-key properly (#302).
- Clarify that ACKs don't get retransmitted, the CONs do (#303).
- Clarify: NON is like separate for CON (#304).
- Don't use decimal response codes, keep the 3+5 structure throughout (#305).
- RFC 2119 usage in 4.5 (#306) and 8.2 (#307).
- Ensure all protocol reactions to reserved or prohibited values are defined (#308).
- URI matching rules may be scheme specific (#309).
- Don't dally beyond MAX_TRANSMIT_SPAN during retransmission (#310).
- More about selecting a token length for anti-spoofing (#311).
- Discuss spoofing ACKs (#312).
- Qualify partial discard strategy implementation note as UDP only (#313).
- Explicitly point out that UDP and DTLS don't mix (#314).
- Point out security consideration re URIs and access control (#315).
- Point to RFC5280 section 6 (#316).

- Add a paragraph about cert status checking (#317).
- RSA is out, ECDHE is in for cert-with-PSK, too (#318).
- Point out that requests and responses don't always come in pairs (#319).
- Clarify when there is a need for Unicode normalization (#320).
- Point out that Uri-Host doesn't handle user-part (#321).
- Clarify the use of non-FQDN Authority Names in certificates.
- Numerous editorial improvements and clarifications.

Changes from ietf-14 to ietf-15: Address comments from IETF last-call, mostly implementation notes and editorial improvements. These should not impact interoperability.

- Clarify bytes/characters and UTF-8/ASCII in "Decomposing URIs into Options" (#282).
- Make reference to ECC/CCM DTLS ciphersuite normative (#286).
- Add a quick warning that bytewise scanning for a payload marker is not a good idea (#287).
- Make reference to PROBING_RATE explicit for saturation discussion (#288).
- Mention PROCESSING_DELAY when discussion piggy-backing (#290).
- Various editorial nits: Clarify use of noun "service" (#283), Reference terminology from lwig-terminology (#284), make reference to HTTP terms more explicit (#285), add a forward reference to 5.9.2.9 (#289), 8 kbit/s is not "conservative" (#291).
- Add description of resource depletion attack (#292).
- Add description of DoS attack on congestion control (#293).
- Add discussion of using non-trivial token for protecting against hijacking (#294).
- Clarify implementation note about per-destination Message ID generation.

Changed from ietf-13 to ietf-14:

- Made Accept option non-repeatable.
- Clarified that Safe options in a 2.03 Valid response update the cache.
- Clarified that payload sniffing is acceptable only if no Content-Format was supplied.
- Clarified URI examples (Appendix B).
- Numerous editorial improvements and clarifications.

Changed from ietf-12 to ietf-13:

- Simplified message format.
  - Removed the OC (Option Count) field in the CoAP Header.
  - Changed the End-of-Options Marker into the Payload Marker.
  - Changed the format of Options: use 4 bits for option length and delta; insert one or two additional bytes after the option header if necessary.
  - Promoted the Token Option to a field following the CoAP Header.
- Clarified when a payload is a diagnostic payload (#264).
- Moved IPsec discussion to separate draft (#262).
- Added a reference to a separate draft on reverse-proxy URI embedding (#259).
- Clarified the use of ETags and of 2.03 responses (#265, #254, #256).
- Added reserved Location-* numbers and clarified Location-*.
- Added Proxy-Scheme proposal.
- Clarified terms such as content negotiation, selected representation, representation-format, message format error.
- Numerous clarifications and a few bugfixes.

Changed from ietf-11 to ietf-12:

- Extended options to support lengths of up to 1034 bytes (#202).

- Added new Jump mechanism for options and removed Fenceposting (#214).
- Added new IANA option number registration policy (#214).
- Added Proxy Unsafe/Safe and Cache-Key masking to option numbers (#241).
- Re-numbered option numbers to use Unsafe/Safe and Cache-Key compliant numbers (#241).
- Defined NSTART and restricted the value to 1 with a MUST (#215).
- Defined PROBING_RATE and set it to 1 Byte/second (#215).
- Defined DEFAULT_LEISURE (#246).
- Renamed Content-Type into Content-Format, and Media Type registry into Content-Format registry.
- A large number of small editorial changes, clarifications and improvements have been made.

Changed from ietf-10 to ietf-11:

- 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.
- Several more small editorial changes, clarifications and improvements have been made.

Changed from ietf-09 to ietf-10:

- Option deltas are restricted to 0 to 14; the option delta 15 is used exclusively for the end-of-options marker (#239).
- Option numbers that are a multiple of 14 are not reserved, but are required to have an empty default value (#212).
- Fixed misleading language that was introduced in 5.10.2 in coap-07 re Uri-Host and Uri-Port (#208).
- Segments and arguments can have a length of zero characters (#213).
- The Location-* options describe together describe one location. The location is a relative URI, not an "absolute path URI" (#218).

- The value of the Location-Path Option must not be '.' or '..' (#218).
- Added a sentence on constructing URIs from Location-* options (#231).
- Reserved option numbers for future Location-* options (#230).
- Fixed response codes with payload inconsistency (#233).
- Added advice on default values for critical options (#207).
- Clarified use of identifiers in RawPublicKey Mode Provisioning (#222).
- Moved "Securing CoAP" out of the "Security Considerations" (#229).
- Added "All CoAP Nodes" multicast addresses to "IANA Considerations" (#216).
- Over 100 small editorial changes, clarifications and improvements have been made.

Changed from ietf-08 to ietf-09:

- Improved consistency of statements about RST on NON: RST is a valid response to a NON message (#183).
- Clarified that the protocol constants can be configured for specific application environments.
- Added implementation note recommending piggy-backing whenever possible (#182).
- Added a content-encoding column to the media type registry (#181).
- Minor improvements to Appendix D.
- Added text about multicast response suppression (#177).
- Included the new End-of-options Marker (#176).
- Added a reference to draft-ietf-tls-oob-pubkey and updated the RPK text accordingly.

Changed from ietf-07 to ietf-08:

- Clarified matching rules for messages (#175)

- Fixed a bug in Section 8.2.2 on Etags (#168)
- Added an IP address spoofing threat analysis contribution (#167)
- Re-focused the security section on raw public keys (#166)
- Added an 4.06 error to Accept (#165)

Changed from ietf-06 to ietf-07:

- application/link-format added to Media types registration (#160)
- Moved content-type attribute to the document from link-format.
- Added coaps scheme and DTLS-secured CoAP default port (#154)
- Allowed 0-length Content-type options (#150)
- Added congestion control recommendations (#153)
- Improved text on PUT/POST response payloads (#149)
- Added an Accept option for content-negotiation (#163)
- Added If-Match and If-None-Match options (#155)
- Improved Token Option explanation (#147)
- Clarified mandatory to implement security (#156)
- Added first come first server policy for 2-byte Media type codes (#161)
- Clarify matching rules for messages and tokens (#151)
- Changed OPTIONS and TRACE to always return 501 in HTTP-CoAP mapping (#164)

Changed from ietf-05 to ietf-06:

- HTTP mapping section improved with the minimal protocol standard text for CoAP-HTTP and HTTP-CoAP forward proxying (#137).
- Eradicated percent-encoding by including one Uri-Query Option per &-delimited argument in a query.
- Allowed RST message in reply to a NON message with unexpected token (#135).

- Cache Invalidation only happens upon successful responses (#134).
- 50% jitter added to the initial retransmit timer (#142).
- DTLS cipher suites aligned with ZigBee IP, DTLS clarified as default CoAP security mechanism (#138, #139)
- Added a minimal reference to draft-kivinen-ipsecme-ikev2-minimal (#140).
- Clarified the comparison of UTF-8s (#136).
- Minimized the initial media type registry (#101).

Changed from ietf-04 to ietf-05:

- Renamed Immediate into Piggy-backed and Deferred into Separate -- should finally end the confusion on what this is about.
- GET requests now return a 2.05 (Content) response instead of 2.00 (OK) response (#104).
- Added text to allow 2.02 (Deleted) responses in reply to POST requests (#105).
- Improved message deduplication rules (#106).
- Section added on message size implementation considerations (#103).
- Clarification made on human readable error payloads (#109).
- Definition of CoAP methods improved (#108).
- Max-Age removed from requests (#107).
- Clarified uniqueness of tokens (#112).
- Location-Query Option added (#113).
- ETag length set to 1-8 bytes (#123).
- Clarified relation between elective/critical and option numbers (#110).
- Defined when to update Version header field (#111).
- URI scheme registration improved (#102).

- Added review guidelines for new CoAP codes and numbers.

Changes from ietf-03 to ietf-04:

- Major document reorganization (#51, #63, #71, #81).
- Max-age length set to 0-4 bytes (#30).
- Added variable unsigned integer definition (#31).
- Clarification made on human readable error payloads (#50).
- Definition of POST improved (#52).
- Token length changed to 0-8 bytes (#53).
- Section added on multiplexing CoAP, DTLS and STUN (#56).
- Added cross-protocol attack considerations (#61).
- Used new Immediate/Deferred response definitions (#73).
- Improved request/response matching rules (#74).
- Removed unnecessary media types and added recommendations for their use in M2M (#76).
- Response codes changed to base 32 coding, new Y.XX naming (#77).
- References updated as per AD review (#79).
- IANA section completed (#80).
- Proxy-Uri Option added to disambiguate between proxy and non-proxy requests (#82).
- Added text on critical options in cached states (#83).
- HTTP mapping sections improved (#88).
- Added text on reverse proxies (#72).
- Some security text on multicast added (#54).
- Trust model text added to introduction (#58, #60).
- AES-CCM vs. AES-CCB text added (#55).

- Text added about device capabilities (#59).
- DTLS section improvements (#87).
- Caching semantics aligned with RFC2616 (#78).
- Uri-Path Option split into multiple path segments.
- MAX_RETRANSMIT changed to 4 to adjust for RESPONSE_TIME = 2.

Changes from ietf-02 to ietf-03:

- Token Option and related use in asynchronous requests added (#25).
- CoAP specific error codes added (#26).
- Erroring out on unknown critical options changed to a MUST (#27).
- Uri-Query Option added.
- Terminology and definitions of URIs improved.
- Security section completed (#22).

Changes from ietf-01 to ietf-02:

- Sending an error on a critical option clarified (#18).
- Clarification on behavior of PUT and idempotent operations (#19).
- Use of Uri-Authority clarified along with server processing rules; Uri-Scheme Option removed (#20, #23).
- Resource discovery section removed to a separate CoRE Link Format draft (#21).
- Initial security section outline added.

Changes from ietf-00 to ietf-01:

- New cleaner transaction message model and header (#5).
- Removed subscription while being designed (#1).
- Section 2 re-written (#3).
- Text added about use of short URIs (#4).

- Improved header option scheme (#5, #14).
- Date option removed whiled being designed (#6).
- New text for CoAP default port (#7).
- Completed proxying section (#8).
- Completed resource discovery section (#9).
- Completed HTTP mapping section (#10).
- Several new examples added (#11).
- URI split into 3 options (#12).
- MIME type defined for link-format (#13, #16).
- New text on maximum message size (#15).
- Location Option added.

Changes from shelby-01 to ietf-00:

- Removed the TCP binding section, left open for the future.
- Fixed a bug in the example.
- Marked current Sub/Notify as (Experimental) while under WG discussion.
- Fixed maximum datagram size to 1280 for both IPv4 and IPv6 (for CoAP-CoAP proxying to work).
- Temporarily removed the Magic Byte header as TCP is no longer included as a binding.
- Removed the Uri-code Option as different URI encoding schemes are being discussed.
- Changed the rel= field to desc= for resource discovery.
- Changed the maximum message size to 1024 bytes to allow for IP/UDP headers.
- Made the URI slash optimization and method idempotence MUSTs
- Minor editing and bug fixing.

Changes from shelby-00 to shelby-01:

- Unified the message header and added a notify message type.
- Renamed methods with HTTP names and removed the NOTIFY method.
- Added a number of options field to the header.
- Combines the Option Type and Length into an 8-bit field.
- Added the magic byte header.
- Added new ETag Option.
- Added new Date Option.
- Added new Subscription Option.
- Completed the HTTP Code - CoAP Code mapping table appendix.
- Completed the Content-type Identifier appendix and tables.
- Added more simplifications for URI support.
- Initial subscription and discovery sections.
- 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
Email: hartke@tzi.org

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

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

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.

Table of Contents

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 or 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 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:

- 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.

- 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.

- More capable servers such as proxies could support a resource directory by requesting the resource descriptions of other end-points 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

- name is either "href", a link-param name defined in this specification, or any other link-extension name, and

- 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 is 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:

- ?href=/foo matches a link-value that is anchored at /foo
- ?href=/foo* matches a link-value that is anchored at a URI that starts with /foo
- ?foo=bar matches a link value that has a target attribute named foo with the exact value bar
- ?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
- ?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 itsef, 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:

- Registration values MUST be related to the intended purpose of these attributes as described in Section 3.
- 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.
- 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".
- 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:

- Attribute Value:
- Description:
- Reference:
- Notes: [optional]

Registration requests should be sent to the 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 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 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 Beroset, 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:

- Editorial clarifications.
- Examples and explanation for filtering when a target attribute of relation-type contains multiple values.

Changes from ietf-12 to ietf-13:

- Improvements to the new CoRE Parameters registry
- Replaced the Section 4.1 ABNF Query Filter definition with a URI Template (#240)
- Aligned examples with rt= and if= value rules
- Clarified that "href" can not be a link parameter

Changes from ietf-11 to ietf-12:

- Changed "uri" to "href" in the filter query (#200)
- Upgraded all ABNF to RFC5234 (#197)
- Put multiple rt= and if= values in a single attribute (as in rel=) (#199)
- Use the Origin definition (#191)
- Clarified URI fetching rules (#196)
- 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.

- 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: July 3, 2015

K. Hartke
Universitaet Bremen TZI

December 30, 2014

Observing Resources in CoAP
draft-ietf-core-observe-16

Abstract

The Constrained Application Protocol (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 enables CoAP clients 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. The protocol follows a best-effort approach for sending new representations to clients and provides eventual consistency between the state observed by each client and the actual resource state at the 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 July 3, 2015.

Table of Contents

1. Introduction

1.1. Background

The Constrained Application Protocol (CoAP) [RFC7252] is intended to provide RESTful services [REST] not unlike HTTP [RFC7230] 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 [RFC7228].

The model of REST is that of a client exchanging representations of resources with a server, where a representation captures the current or intended state of a resource. The server is the authority for representations of the resources in its namespace. A client interested in the state of a resource initiates a request to the server, the server then returns a response with a representation of the resource 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 from HTTP, such as repeated polling or HTTP long polling [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 for a CoAP client to "observe" a resource on a CoAP server: the client retrieves a representation of the resource and requests this representation be updated by the server as long as the client is interested in the resource.

The protocol keeps the architectural properties of REST. It enables high scalability and efficiency through the support of caches and proxies. There is no intention, though, 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 to an observer, the observer must register separately for all of them.

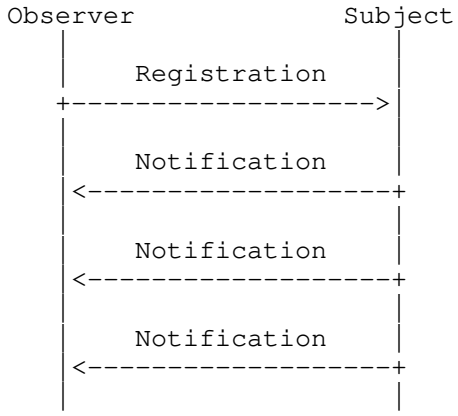


Figure 1: The Observer Design Pattern

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 having a current representation of the resource at any given time.

Registration: A client registers its interest in a resource by initiating 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 in the list of observers of the resource. Each notification is an additional CoAP response sent by the server in reply to the single extended GET request, and includes a complete, updated representation of the new resource state.

Figure 2 below shows an example of a CoAP client registering its interest in a resource and receiving three notifications: the first with the current state upon registration, and then two upon changes to the resource state. Both the registration request and the notifications are identified as such by the presence of the Observe Option defined in this document. In notifications, the Observe Option additionally provides a sequence number for reordering detection. All notifications carry the token specified by the client, so the client can easily correlate them to the request.

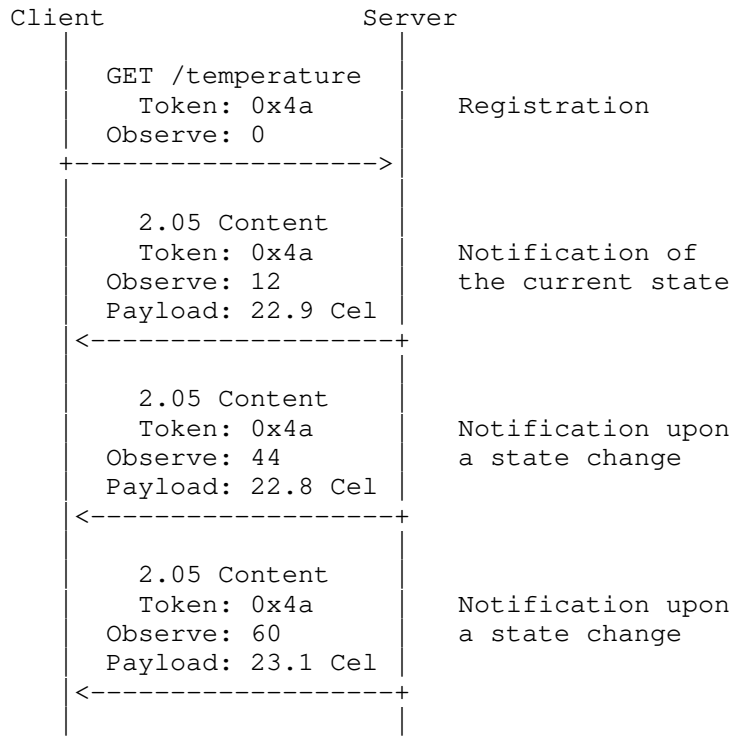


Figure 2: Observing a Resource in CoAP

A client remains on the list of observers as long as the server can determine the client's continued interest in the resource. The server may send a notification in a confirmable CoAP message to request an acknowledgement from the client. When the client deregisters, rejects a notification, or the transmission of a notification times out after several transmission attempts, the client is considered no longer interested in the resource and is removed by the server from the list of observers.

1.3. Consistency Model

While a client is in the list of observers of a resource, the goal of the protocol is to keep the resource state observed by the client as closely in sync with the actual state at the server as possible.

It cannot be avoided that the client and the server become out of sync at times: First, there is always some latency between the change of the resource state and the receipt of the notification. Second, CoAP messages with notifications can get lost, which will cause the client to assume an old state until it receives a new notification.

And third, the server may erroneously come to the conclusion that the client is no longer interested in the resource, which will cause the server to stop sending notifications and the client to assume an old state until it eventually registers its interest again.

The protocol addresses this issue as follows:

- It follows a best-effort approach for sending the current representation to the client after a state change: Clients should see the new state after a state change as soon as possible, and they should see as many states as possible. This is limited by congestion control, however, so a client cannot rely on observing every single state that a resource might go through.

- It labels notifications with a maximum duration up to which it is acceptable for the observed state and the actual state to be out of sync. When the age of the notification received reaches this limit, the client cannot use the enclosed representation until it receives a new notification.

- It is designed on the principle of eventual consistency: The protocol guarantees that, if the resource does not undergo a new change in state, eventually all registered observers will have a current representation of the latest resource state.

1.4. Observable Resources

A CoAP server is the authority for determining under what conditions resources change their state and thus when observers are notified of new resource states. The protocol does not offer explicit means for setting up triggers or thresholds; it is up to the server to expose observable resources that change their state in a way that is useful in the application context.

For example, a CoAP server with an attached temperature sensor could expose one or more of the following resources:

- <coap://server/temperature>, which changes its state every few seconds to a current reading of the temperature sensor;

- <coap://server/temperature/felt>, which changes its state to "COLD" whenever the temperature reading drops below a certain pre-configured threshold, and to "WARM" whenever the reading exceeds a second, slightly higher threshold;

- <coap://server/temperature/critical?above=42>, which changes its state based on the client-specified parameter value: every few seconds to the current temperature reading if the temperature

exceeds the threshold, or to "OK" when the reading drops below;

- <coap://server/?query=select+avg(temperature)+from+Sensor.window:time(30sec)>, which accepts expressions of arbitrary complexity and changes its state accordingly.

Thus, by designing CoAP resources that change their state on certain conditions, it is possible to update the client only when these conditions occur instead of supplying it continuously with raw sensor data. By parameterizing resources, this is not limited to conditions defined by the server, but can be extended to arbitrarily complex queries specified by the client. The application designer therefore can choose exactly the right level of complexity for the application envisioned and devices involved, and is not constrained to a "one size fits all" mechanism built into the protocol.

1.5. 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 RFC 2119 [RFC2119].

2. The Observe Option

The Observe Option has the following properties. Its meaning depends on whether it is included in a GET request or in a response.

No.	C	U	N	R	Name	Format	Length	Default
6		x	-		Observe	uint	0-3 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Table 1: The Observe Option

When included in a GET request, the Observe Option extends the GET method so it does not only retrieve a current representation of the target resource, but also requests the server to add or remove an entry in the list of observers of the resource, depending on the option value. The list entry consists of the client endpoint and the token specified by the client in the request. Possible values are:

0 (register) adds the entry to the list, if not present;

1 (deregister) removes the entry from the list, if present.

The Observe Option is not critical for processing the request. If the server is unwilling or unable to add a new entry to the list of observers, then the request falls back to a normal GET request, and the response does not include the Observe Option.

The Observe Option is not part of the cache-key: a cacheable response obtained with an Observe Option in the request can be used to satisfy a request without an Observe Option, and vice versa. When a stored response with an Observe Option is used to satisfy a normal GET request, the option MUST be removed before the response is returned.

When included in a response, the Observe Option identifies the message as a notification. This implies that a matching entry exists in the list of observers and that the server will notify the client of changes to the resource state. The option value is a sequence number for reordering detection (see Section 3.4 and Section 4.4).

The value of the Observe Option is encoded as an unsigned integer in network byte order using a variable number of bytes ('uint' option format); see Section 3.2 of RFC 7252 [RFC7252].

3. Client-side Requirements

3.1. Request

A client registers its interest in a resource by issuing a GET request with an Observe Option set to 0 (register). If the server returns a 2.xx response that includes an Observe Option as well, the server has successfully added an entry with the client endpoint and request token to the list of observers of the target resource and the client will be notified of changes to the resource state.

Like a fresh response can be used to satisfy a request without contacting the server, the stream of updates resulting from one observation request can be used to satisfy another (observation or normal GET) request if the target resource is the same. A client MUST aggregate such requests and MUST NOT register more than once for the same target resource. The target resource is identified by all options in the request that are part of the cache-key. This includes, for example, the full request URI and the Accept Option.

3.2. Notifications

Notifications are additional responses sent by the server in reply to the single extended GET request that created the registration. Each notification includes the token specified by the client in the request. The only difference between a notification and a normal response is the presence of the Observe Option.

Notifications typically have a 2.05 (Content) response code. They include an Observe Option with a sequence number for reordering detection (see Section 3.4), and a payload in the same Content-Format as the initial response. If the client included one or more ETag Options in the GET request (see Section 3.3), notifications can have a 2.03 (Valid) response code rather than a 2.05 (Content) response code. Such notifications include an Observe Option with a sequence number but no payload.

In the event that the resource changes in a way that would cause a normal GET request at that time to return a non-2.xx response (for example, when the resource is deleted), the server sends a notification with an appropriate response code (such as 4.04 Not Found) and removes the client's entry from the list of observers of the resource. Non-2.xx responses do not include an Observe Option.

3.3. Caching

As notifications are just additional responses to a GET request, notifications partake in caching as defined in Section 5.6 of RFC 7252 [RFC7252]. Both the freshness model and the validation model are supported.

3.3.1. Freshness

A client MAY store a notification like a response in its cache and use a stored notification that is fresh without contacting the server. Like a response, a notification is considered fresh while its age is not greater than the value indicated by the Max-Age Option (and no newer notification/response has been received).

The server will do its best to keep the resource state observed by the client as closely in sync with the actual state as possible. However, a client cannot rely on observing every single state that a resource might go through. For example, if the network is congested or the state changes more frequently than the network can handle, the server can skip notifications for any number of intermediate states.

The server uses the Max-Age Option to indicate an age up to which it is acceptable that the observed state and the actual state are inconsistent. If the age of the latest notification becomes greater than its indicated Max-Age, then the client MUST NOT assume that the enclosed representation reflects the actual resource state.

To make sure it has a current representation and/or to re-register its interest in a resource, a client MAY issue a new GET request with the same token as the original at any time. All options MUST be identical to those in the original request, except for the set of

ETag Options. It is RECOMMENDED that the client does not issue the request while it still has a fresh notification/response for the resource in its cache. Additionally, the client SHOULD at least wait for a random amount of time between 5 and 15 seconds after Max-Age expired to reduce collisions with other clients.

3.3.2. Validation

When a client has one or more notifications stored in its cache for a resource, it can use the ETag Option in the GET request to give the server an opportunity to select a stored notification to be used.

The client MAY include an ETag Option for each stored response that is applicable in the GET request. Whenever the observed resource changes 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.

A client implementation needs to keep all candidate responses in its cache until it is no longer interested in the target resource or it re-registers with a new set of entity-tags.

3.4. Reordering

Messages with notifications can arrive in a different order than they were sent. Since the goal is to keep the observed state as closely in sync with the actual state as possible, a client MUST consider the notification that was sent most recently as the freshest, regardless of the order of arrival.

To provide an order among notifications for the client, the server sets the value of the Observe Option in each notification to the 24 least-significant bits of a strictly increasing sequence number. An incoming notification was sent more recently than the freshest notification so far when one of the following conditions is met:

```
(V1 < V2 and V2 - V1 < 2^23) or
(V1 > V2 and V1 - V2 > 2^23) or
(T2 > T1 + 128 seconds)
```

where V1 is the value of the Observe Option in the freshest notification so far, V2 the value of the Observe Option in the incoming notification, T1 a client-local timestamp for the freshest notification so far, and T2 a client-local timestamp for the incoming notification.

Design Note: The first two conditions verify that V1 is less than V2 in 24-bit serial number arithmetic [RFC1982]. The third condition ensures that, if the server is generating serial numbers based on a local clock, the time elapsed between the two incoming messages is not so large that the difference between V1 and V2 has become larger than the largest integer that it is meaningful to add to a 24-bit serial number; in other words, after 128 seconds have elapsed without any notification, a client does not need to check the sequence numbers to assume that an incoming notification was sent more recently than the freshest notification it has received so far.

The duration of 128 seconds was chosen as a nice round number greater than MAX_LATENCY (Section 4.8.2 of RFC 7252 [RFC7252]).

3.5. Transmission

A notification can be confirmable or non-confirmable, i.e., it can be sent in a confirmable or a non-confirmable message. The message type used for a notification is independent of the type used for the request and of any previous notification.

If a client does not recognize the token in a confirmable notification, it MUST NOT acknowledge the message and SHOULD reject it with a Reset message; otherwise, the client MUST acknowledge the message as usual. In the case of a non-confirmable notification, rejecting the message with a Reset message is OPTIONAL.

An acknowledgement message 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 the associated entry from the list of observers.

3.6. Cancellation

A client that is no longer interested in receiving notifications for a resource can simply "forget" the observation. When the server then sends the next notification, the client will not recognize the token in the message and thus will return a Reset message. This causes the server to remove the associated entry from the list of observers. The entries in lists of observers are effectively "garbage collected" by the server.

Implementation Note: Due to potential message loss, the Reset message may not reach the server. The client may therefore have to reject multiple notifications, each with one Reset message, until the server finally removes the associated entry from the list of observers and stops sending notifications.

In some circumstances, it may be desirable to cancel an observation and release the resources allocated by the server to it more eagerly. In this case, a client MAY explicitly deregister by issuing a GET request which has the Token field set to the token of the observation to be cancelled and includes an Observe Option with the value set to 1 (deregister). All other options MUST be identical to those in the registration request, except for the set of ETag Options. When the server receives such a request, it will remove any matching entry from the list of observers and process the GET request as usual.

4. Server-side Requirements

4.1. Request

A GET request with an Observe Option set to 0 (register) requests the server not only to return a current representation of the target resource, but also to add the client to the list of observers of that resource. Upon success, the server returns a current representation of the resource and MUST keep this representation updated (as described in Section 1.3) as long as the client is on the list of observers.

The entry in the list of observers is keyed by the client endpoint and the token specified by the client in the request. If an entry with a matching endpoint/token pair is already present in the list (which, for example, happens when the client wishes to reinforce its interest in a resource), the server MUST NOT add a new entry but MUST replace or update the existing one.

A server that is unable or unwilling to add a new entry to the list of observers of a resource MAY silently ignore the registration request 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 and, e.g., needs to poll the resource for its state instead.

If the Observe Option in a GET request is set to 1 (deregister), then the server MUST remove any existing entry with a matching endpoint/ token pair from the list of observers and process the GET request as usual. The resulting response MUST NOT include an Observe Option.

4.2. Notifications

A client is notified of changes to the resource state by additional responses sent by the server in reply to the GET request. Each such notification response (including the initial response) MUST echo the token specified by the client in the GET request. If there are multiple entries in the list of observers, the order in which the clients 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 at that time to return a non-2.xx response (for example, when the resource is deleted), the server SHOULD notify the client by sending a notification with an appropriate response code (such as 4.04 Not Found) and subsequently MUST remove the associated entry from the list of observers of the resource.

The Content-Format specified in a 2.xx 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 in this format, it SHOULD send a notification with a 4.06 (Not Acceptable) response code and subsequently MUST remove the associated entry from the list of observers of the resource.

A 2.xx notification MUST include an Observe Option with a sequence number as specified in Section 4.4 below; a non-2.xx notification MUST NOT include an Observe Option.

4.3. Caching

As notifications are just additional responses sent by the server in reply to a GET request, they are subject to caching as defined in Section 5.6 of RFC 7252 [RFC7252].

4.3.1. Freshness

After returning the initial response, the server MUST keep the resource state that is observed by the client as closely in sync with the actual resource state as possible.

Since becoming out of sync at times cannot be avoided, the server MUST indicate for each representation an age up to which it is acceptable that the observed state and the actual state are inconsistent. This age is application-dependent and MUST be specified in notifications using the Max-Age Option.

When the resource does not change and the client has a current representation, the server does not need to send a notification. However, if the client does not receive a notification, the client cannot tell if the observed state and the actual state are still in sync. Thus, when the age of the latest notification becomes greater than its indicated Max-Age, the client no longer has a usable representation of the resource state. The server MAY wish to prevent that by sending a new notification with the unchanged representation and a new Max-Age just before the Max-Age indicated earlier expires.

4.3.2. Validation

A client can include a set of entity-tags in its request using the ETag Option. When a 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.

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 of each notification it sends to the 24 least-significant bits of a strictly increasing sequence number. The sequence number MAY start at any value and MUST NOT increase so fast that it increases by more than 2^23 within less than 256 seconds.

The sequence number selected for a notification MUST be greater than that of any preceding notification sent to the same client with the same token for the same resource. The value of the Observe Option MUST be current at the time of transmission; if a notification is retransmitted, the server MUST update the value of the option to the sequence number that is current at that time before retransmission.

Implementation Note: A simple implementation that satisfies the requirements is to obtain a timestamp from a local clock. The sequence number then is the timestamp in ticks, where 1 tick = (256 seconds)/(2^23) = 30.52 microseconds. It is not necessary that the clock reflects the current time/date.

Another valid implementation is to store a 24-bit unsigned integer variable per resource and increment this variable each time the resource undergoes a change of state (provided that the resource changes its state less than 2^23 times in the first 256 seconds after every state change). This removes the need to update the value of the Observe Option on retransmission when the resource

state did not change.

Design Note: The choice of a 24-bit option value and a time span of 256 seconds theoretically allows for a notification rate of up to 65536 notifications per second. Constrained nodes often have rather imprecise clocks, though, and inaccuracies of the client and server side may cancel out or add in effect. Therefore, the maximum notification rate is reduced to 32768 notifications per second. This is still well beyond the highest known design objective of around 1 kHz (most CoAP applications will be several orders of magnitude below that), but allows total clock inaccuracies of up to -50/+100 %.

4.5. Transmission

A notification can be sent in 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.

A server MAY choose to skip sending a notification if it knows that it will send another notification soon, for example, when the state of a resource is changing frequently. It also MAY choose to send more than one notification for the same resource state. However, above all, the server MUST ensure that a client in the list of observers of a resource eventually observes the latest state if the resource does not undergo a new change in state.

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 by repeating the last notification in a confirmable message when the burst is over.

The client's acknowledgement of a confirmable notification signals that the client is interested in receiving further notifications. If a client rejects a confirmable or non-confirmable notification with a Reset message, or if the last attempt to retransmit a confirmable notification times out, then the client is considered no longer interested and the server MUST remove the associated entry from the list of observers.

Implementation Note: To properly process a Reset message that rejects a non-confirmable notification, a server needs to remember the message IDs of the non-confirmable notifications it sends. This may be challenging for a server with constrained resources. However, since Reset messages are transmitted unreliably, the client must be prepared that its Reset messages aren't received by the server. A server thus can always pretend that a Reset message rejecting a non-confirmable notification was lost. If a server does this, it could accelerate cancellation by sending the following notifications to that client in confirmable messages.

A server that transmits notifications mostly in non-confirmable messages MUST send a notification in a confirmable message instead of a non-confirmable message at least every 24 hours. This prevents a client that went away or is no longer interested from remaining in the list of observers indefinitely.

4.5.1. Congestion Control

Basic congestion control for CoAP is provided by the exponential back-off mechanism in Section 4.2 of RFC 7252 [RFC7252] and the limitations in Section 4.7 of RFC 7252 [RFC7252]. However, CoAP places the responsibility of congestion control for simple request/response interactions only on the clients: rate limiting request transmission implicitly controls the transmission of the responses. When a single request yields a potentially infinite number of notifications, additional responsibility needs to be placed on the server.

In order not to cause congestion, servers MUST strictly limit the number of simultaneous outstanding notifications/responses that they transmit to a given client to NSTART (1 by default; see Section 4.7 of RFC 7252 [RFC7252]). An outstanding notification/response is either a confirmable message for which an acknowledgement has not yet been received and whose last retransmission attempt has not yet timed out, or a non-confirmable message for which the waiting time that results from the following rate limiting rules has not yet elapsed.

The server SHOULD NOT send more than one non-confirmable notification per round-trip time (RTT) to a client on average. If the server cannot maintain an RTT estimate for a client, it SHOULD NOT send more than one non-confirmable notification every 3 seconds, and SHOULD use an even less aggressive rate when possible (see also Section 3.1.2 of RFC 5405 [RFC5405]).

Further congestion control optimizations and considerations are expected in the future with advanced CoAP congestion control mechanisms.

4.5.2. Advanced Transmission

The state of an observed resource may change while the number of the number of simultaneous outstanding notifications/responses to a client on the list of observers is greater than or equal to NSTART. In this case, the server cannot notify the client of the new resource state immediately but has to wait for an outstanding notification/ response to complete first.

If there exists an outstanding notification/response that the server transmits to the client and that pertains to the changed resource, then it is desirable for the server to stop working towards getting the representation of the old resource state to the client, and to start transmitting the current representation to the client instead, so the resource state observed by the client stays closer in sync with the actual state at the server.

For this purpose, the server MAY optimize the transmission process by aborting the transmission of the old notification (but not before the current transmission attempt completed) and starting a new transmission for the new notification (but with the retransmission timer and counter of the aborted transmission retained).

In more detail, a server MAY supersede an outstanding transmission that pertains to an observation as follows:

1. Wait for the current (re-)transmission attempt to be acknowledged, rejected or to time out (confirmable transmission); or wait for the waiting time to elapse or the transmission to be rejected (non-confirmable transmission).

2. If the transmission is rejected or it was the last attempt to retransmit a notification, remove the associated entry from the list of observers of the observed resource.

3. If the entry is still in the list of observers, start to transmit a new notification with a representation of the current resource state. Should the resource have changed its state more than once in the meantime, the notifications for the intermediate states are silently skipped.

4. The new notification is transmitted with a new Message ID and the following transmission parameters: If the previous (re-)transmission attempt timed out, retain its transmission parameters, increment the retransmission counter and double the timeout; otherwise, initialize the transmission parameters as usual (see Section 4.2 of RFC 7252 [RFC7252]).

It is possible that the server later receives an acknowledgement for a confirmable notification that it superseded this way. Even though this does not signal consistency, it is valuable in that it signals the client's further interest in the resource. The server therefore should avoid inadvertently removing the associated entry from the list of observers.

5. Intermediaries

A client may be interested in a resource in the namespace of a server that is reached through a chain of one or more CoAP intermediaries. In this case, the client registers its interest with the first intermediary towards the server, acting as if it was communicating with the 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 to keep the representation updated upon changes to the resource state, 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 towards the server. If the response returned by the next hop doesn't include an Observe Option, the intermediary MAY resort to polling the next hop or MAY itself return a response without an Observe Option.

The communication between each pair of hops is independent; each hop in the server role MUST determine individually how many notifications to send, of which message type, and so on. Each hop MUST generate its own values for the Observe Option in notifications, and MUST set the value of the Max-Age Option according to the age of the local current representation.

If two or more clients have registered their interest in a resource with an intermediary, the intermediary MUST register itself only once with the next hop and fan out the notifications it receives to all registered clients. This relieves the next hop from sending the same notifications multiple times and thus enables scalability.

An intermediary is not required to act on behalf of a client to observe a resource; an intermediary MAY observe a resource, for example, just to keep its own cache up to date.

See Appendix A.2 for examples.

6. Web Linking

A web link [RFC5988] to a resource accessible over CoAP (for example, in a link-format document [RFC6690]) MAY include the target attribute "obs".

The "obs" attribute, when present, is a hint indicating that the destination of a link is useful for observation and thus, for example, should have a suitable graphical representation in a user interface. Note that this is only a hint; it is not a promise that the Observe Option can actually be used to perform the observation. A client may need to resort to polling the resource if the Observe Option is not returned in the response to the GET request.

A value MUST NOT be given for the "obs" attribute; any present value MUST be ignored by parsers. The "obs" attribute MUST NOT appear more than once in a given link-value; occurrences after the first MUST be ignored by parsers.

7. Security Considerations

The security considerations in Section 11 of the CoAP specification [RFC7252] apply.

Observing resources can dramatically increase the negative effects of amplification attacks. That is, not only can notifications messages be much larger than the request message, but the nature of the protocol can cause a significant number of notifications to be generated. Without client authentication, a server therefore MUST 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.)

The protocol follows a best-effort approach for keeping the state observed by a client and the actual resource state at a server in sync. This may have the client and the server become out of sync at times. Depending on the sensitivity of the observed resource, operating on an old state might be a security threat. The client therefore must be careful not to use a representation after its Max-Age expires, and the server must set the Max-Age Option to a sensible value.

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 apply

access controls to 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.

Resources can be observed over DTLS-secured CoAP using any of the security modes described in Section 9 of RFC 7252. The use of DTLS is indicated by the "coaps" URI scheme. All notifications resulting from a GET request with an Observe Option MUST be returned within the same epoch of the same connection as the request.

8. IANA Considerations

The following entry is added to the CoAP Option Numbers registry:

Number	Name	Reference
6	Observe	[RFCXXXX]

[Note to RFC Editor: Please replace XXXX with the RFC number of this specification.]

9. 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 A. Bigot, Angelo P. Castellani, Gilbert Clark, Esko Dijk, Thomas Fossati, Brian Frank, Bert Greevenbosch, Jeroen Hoebeke, Cullen Jennings, Matthias Kovatsch, Barry Leiba, Salvatore Loreto, Charles Palmer, Akbar Rahman, Zach Shelby, and Floris Van den Abeele for helpful comments and discussions that have shaped the document.

This work was supported in part by Klaus Tschira Foundation, Intel, Cisco, and Nokia.

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.

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

10.2. 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.

[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>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[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.

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

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

Appendix A. Examples

A.1. Client/Server Examples

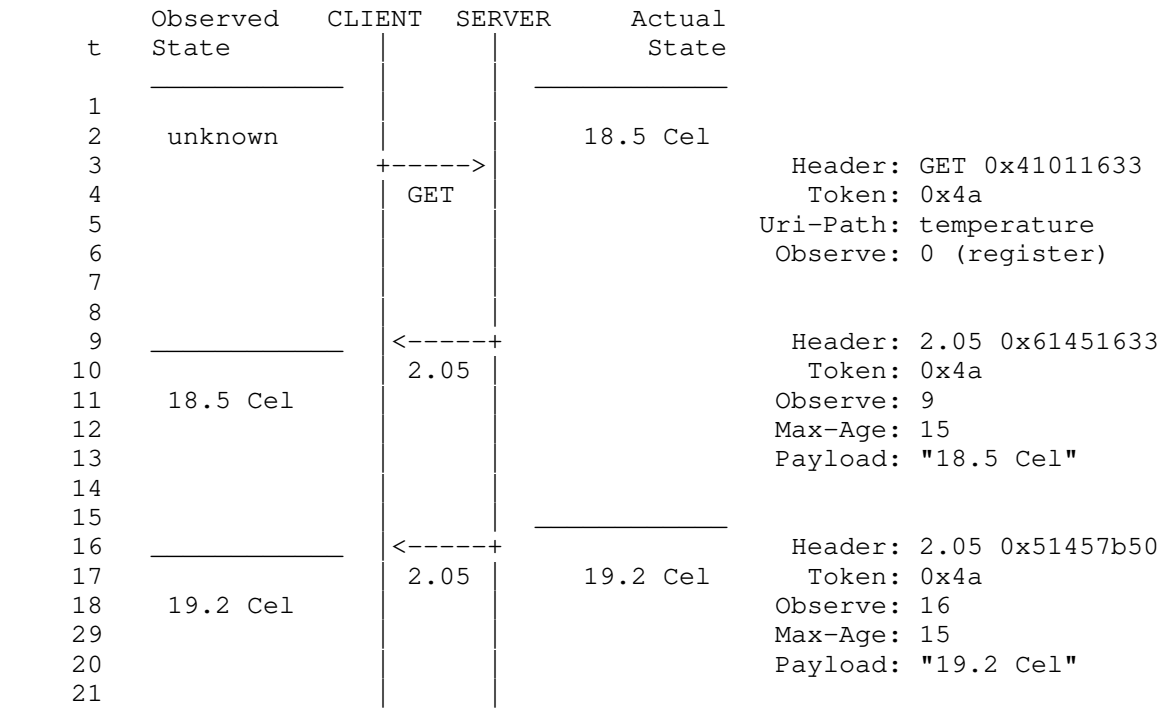


Figure 3: A client registers and receives one notification of the current state and one of a new state upon a state change

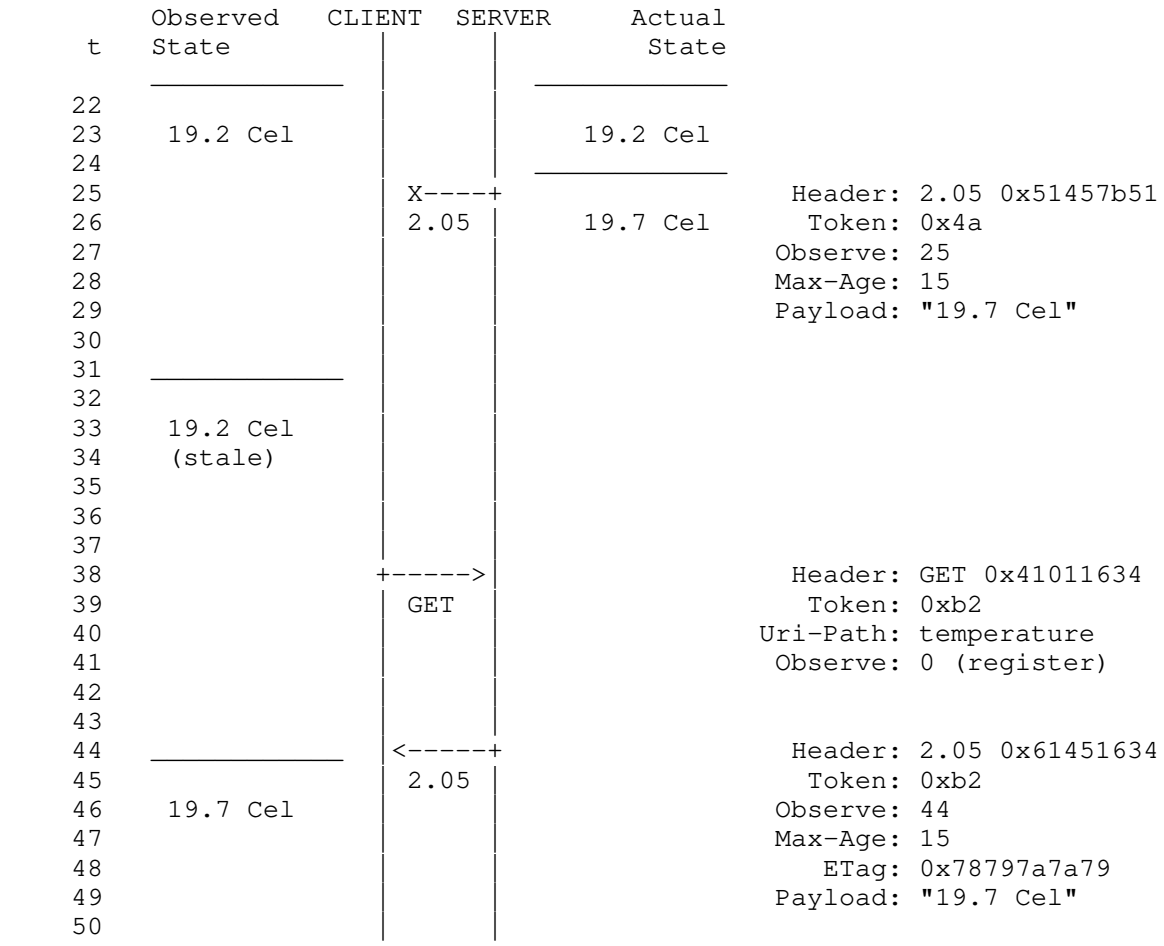


Figure 4: The client re-registers after Max-Age ends

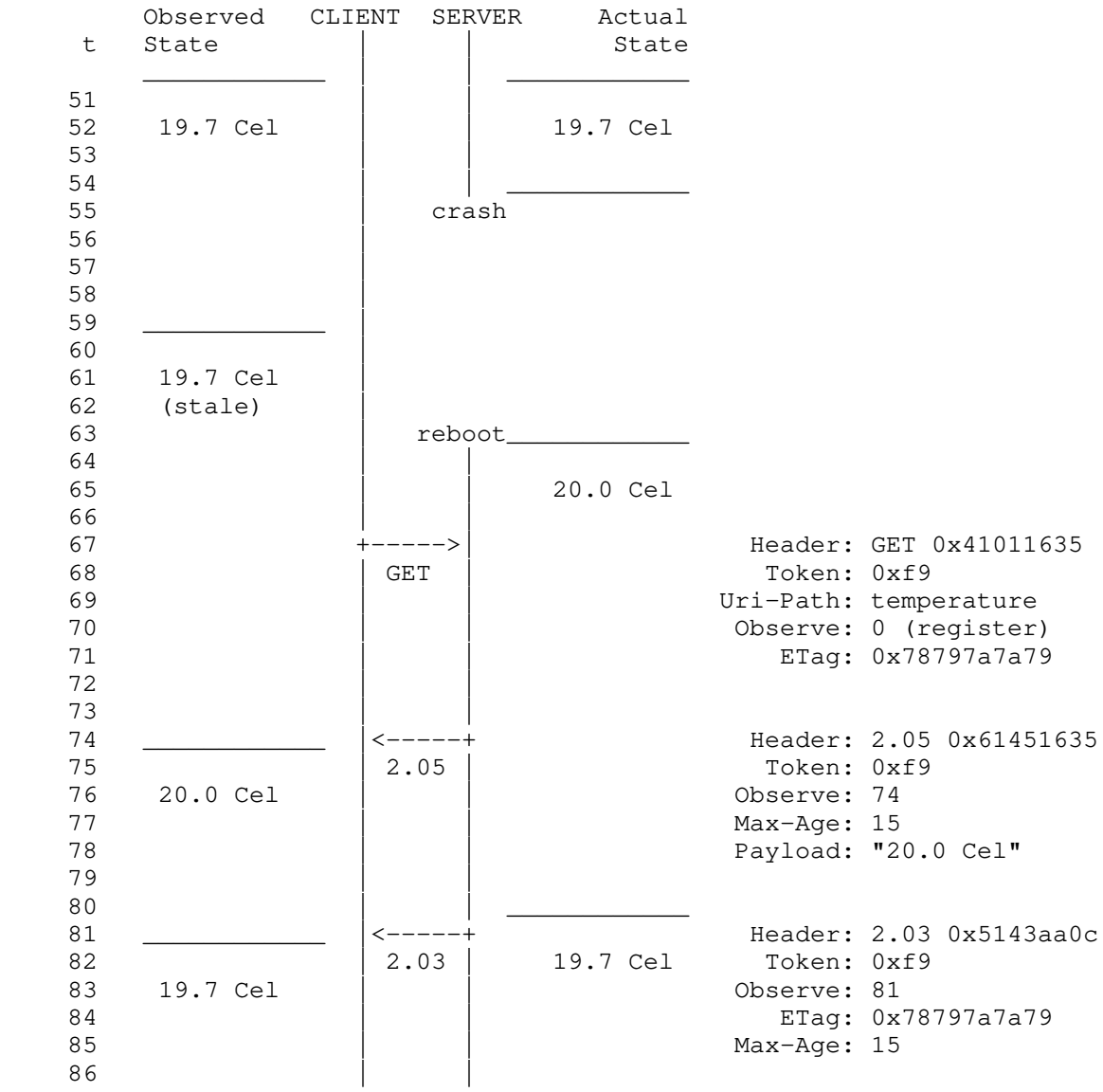


Figure 5: The client re-registers and gives the server the opportunity to select a stored response

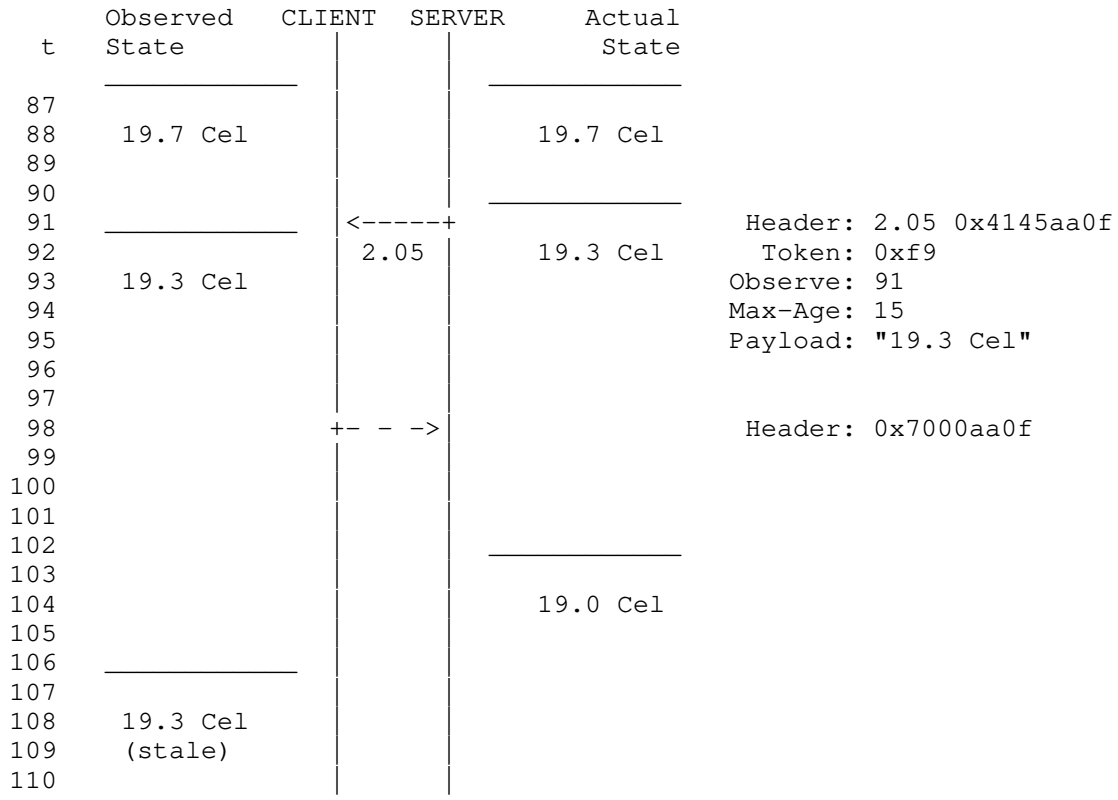


Figure 6: The client rejects a notification and thereby cancels the observation

A.2. Proxy Examples

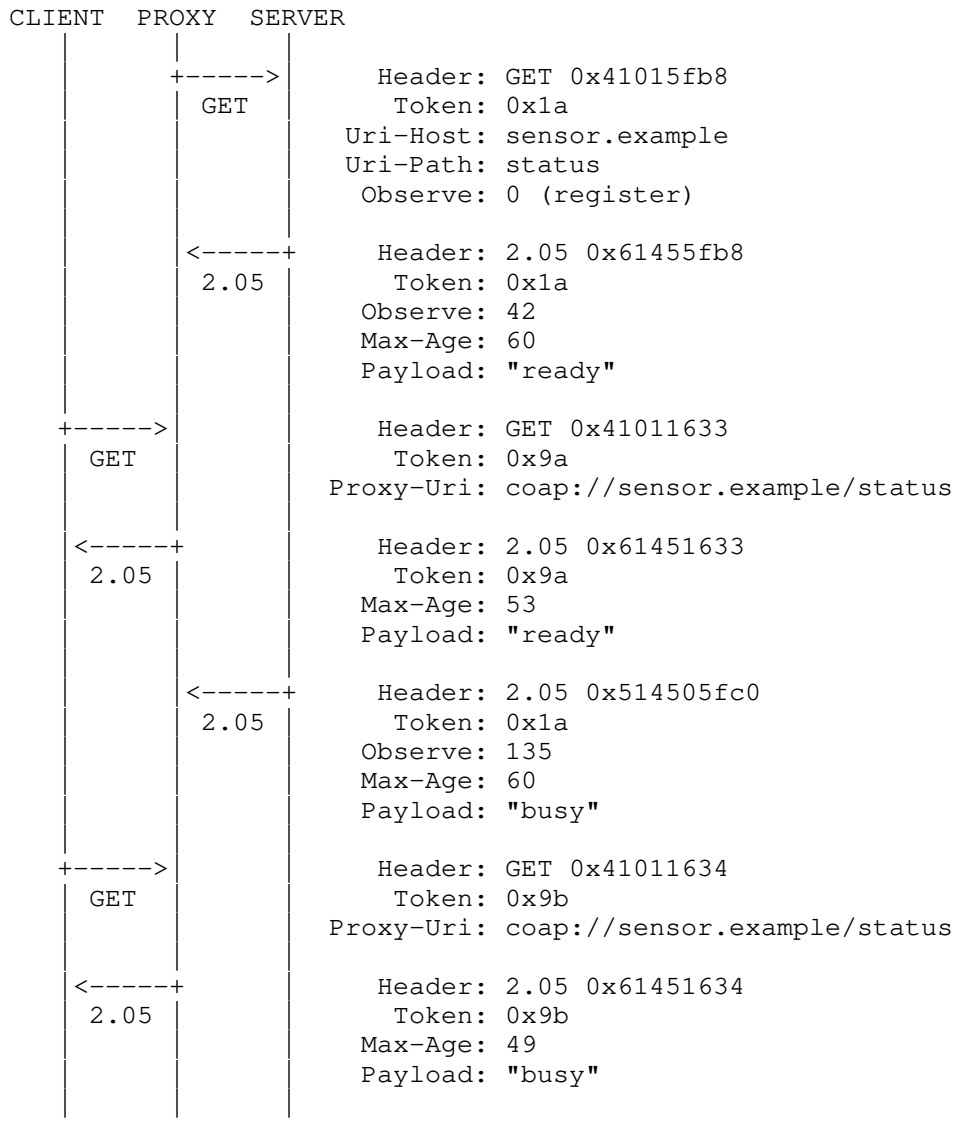


Figure 7: A proxy observes a resource to keep its cache up to date

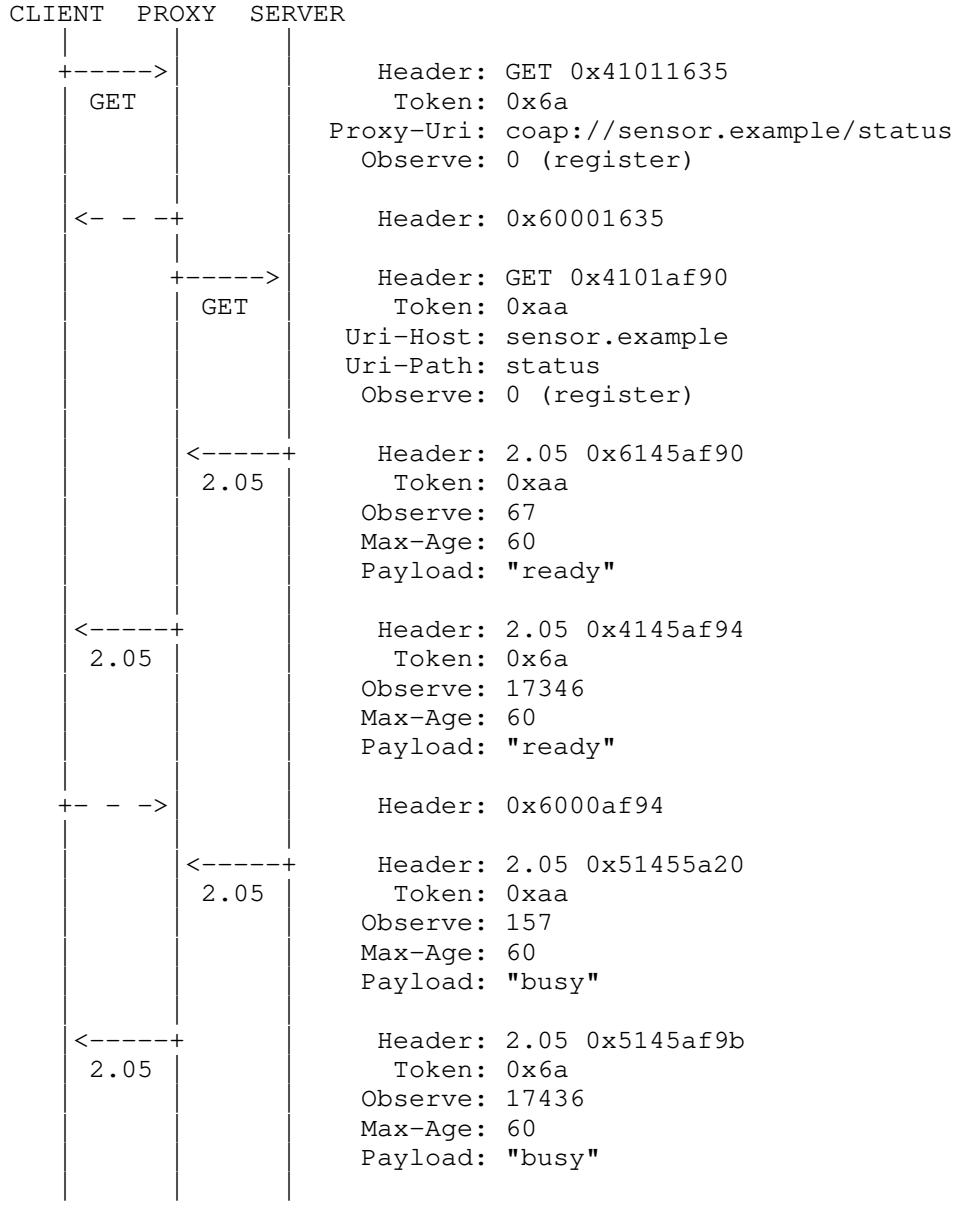


Figure 8: A client observes a resource through a proxy

Appendix B. Changelog

[Note to RFC Editor: Please remove this section before publication.]

Changes from ietf-15 to ietf-16:

- Clarified several points based on AD, GenART, IESG, and Secdir reviews.

Changes from ietf-14 to ietf-15:

- Clarified several points based on AD, GenART, IESG, and Secdir reviews.

Changes from ietf-13 to ietf-14:

- Updated references.

Changes from ietf-12 to ietf-13:

- Extended the Observe Option in requests to not only add but also remove an entry in the list of observers, depending on the option value.

  Note: The value of the Observe Option in a registration request may now be any sequence of bytes that encodes the unsigned integer 0, i.e., 0x'', 0x'00', 0x'00 00' or 0x'00 00 00'.

- Removed the 7.31 Code for cancellation.

Changes from ietf-11 to ietf-12:

- Introduced the 7.31 Code to request the cancellation of a pending request.

- Made the algorithm for superseding an outstanding transmission OPTIONAL.

- Clarified that the entry in the list of observers is removed if the client fails to acknowledge a confirmable notification before the last retransmission attempt times out (#350).

- Simplified the text on cancellation (#352) and the handling of Reset messages (#353).

Changes from ietf-10 to ietf-11:

- Pointed out that client and server clocks may differ in their realization of the SI second, and added robustness to the existing reordering scheme by reducing the maximum notification rate to 32768 notifications per second (#341).

Changes from ietf-09 to ietf-10:

- Required consistent sequence numbers across requests (#333).
- Clarified that a server needs to update the entry in the list of observers instead of adding a new entry if the endpoint/token pair is already present.
- Allowed that a client uses a token that is currently in use to ensure that it's still in the list of observers. This is possible because sequence numbers are now consistent across requests and servers won't add a new entry for the same token.
- Improved text on the transmission of non-confirmable notifications to match Section 3.1.2 of RFC 5405 more closely.
- Updated examples to use UCUM units.
- Moved Appendix B into the introduction.

Changes from ietf-08 to ietf-09:

- Removed the side effects of requests on existing observations. This includes removing that
  - the client can use a GET request to cancel an observation;
  - the server updates the entry in the list of observers instead of adding a new entry if the client is already present (#258, #281).
- Clarified that a resource (and hence an observation relationship) is identified by the request options that are part of the Cache-Key (#258).
- Clarified that a non-2.xx notification MUST NOT include an Observe Option.
- Moved block-wise transfer of notifications to [I-D.ietf-core-block].

Changes from ietf-07 to ietf-08:

- Expanded text on transmitting a notification while a previous transmission is pending (#242).
- Changed reordering detection to use a fixed time span of 128 seconds instead of EXCHANGE_LIFETIME (#276).
- Removed the use of the freshness model to determine if the client is still on the list of observers. This includes removing that
  - the client assumes that it has been removed from the list of observers when Max-Age ends;
  - the server sets the Max-Age Option of a notification to a value that indicates when the server will send the next notification;
  - the server uses a number of retransmit attempts such that removing a client from the list of observers before Max-Age ends is avoided (#235);
  - the server may remove the client from all lists of observers when the transmission of a confirmable notification ultimately times out.
- Changed that an unrecognized critical option in a request must actually have no effect on the state of any observation relationship to any resource, as the option could lead to a different target resource.
- Clarified that client implementations must be prepared to receive each notification equally as a confirmable or a non-confirmable message, regardless of the message type of the request and of any previous notification.
- Added a requirement for sending a confirmable notification at least every 24 hours before continuing with non-confirmable notifications (#221).
- Added congestion control considerations from [I-D.bormann-core-congestion-control-02].
- Recommended that the client waits for a randomized time after the freshness of the latest notification expired before re-registering. This prevents that multiple clients observing a resource perform a GET request at the same time when the need to re-register arises.
- Changed reordering detection from 'MAY' to 'SHOULD', as the goal of the protocol (to keep the observed state as closely in sync

with the actual state as possible) is not optional.

- Fixed the length of the Observe Option (3 bytes) in the table in Section 2.

- Replaced the 'x' in the No-Cache-Key column in the table in Section 2 with a '-', as the Observe Option doesn't have the No-Cache-Key flag set, even though it is not part of the cache key.

- Updated examples.

Changes from ietf-06 to ietf-07:

- Moved to 24-bit sequence numbers to allow for up to 15000 notifications per second per client and resource (#217).

- Re-numbered option number to use Unsafe/Safe and Cache-Key compliant numbers (#241).

- Clarified how to react to a Reset message that is sent in reply to a non-confirmable notification (#225).

- Clarified the semantics of the "obs" link target attribute (#236).

Changes from ietf-05 to ietf-06:

- Improved abstract and introduction to say that the protocol is about best effort and eventual consistency (#219).

- Clarified that the value of the Observe Option in a request must have zero length.

- Added requirement that the sequence number must be updated each time a server retransmits a notification.

- Clarified that a server must remove a client from the list of observers when it receives a GET request with an unrecognized critical option.

- Updated the text to use the endpoint concept from [I-D.ietf-core-coap] (#224).

- Improved the reordering text (#223).

Changes from ietf-04 to ietf-05:

- 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).

- Relaxed requirements when sending a Reset message in reply to non-confirmable notifications.
- Added an implementation note about careless GET requests (#184).
- Updated examples.

Changes from ietf-03 to ietf-04:

- Removed the "Max-OFE" Option.
- Allowed a Reset message in reply to non-confirmable notifications.
- Added a section on cancellation.
- Updated examples.

Changes from ietf-02 to ietf-03:

- Separated client-side and server-side requirements.
- 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).
- Simplified the text on message reordering (#129).
- Clarified requirements for intermediaries.
- Clarified the combination of blockwise transfers with notifications (#172).
- Updated examples to show how the state observed by the client becomes eventually consistent with the actual state on the server.
- Added examples for parameterization of observable resource.

Changes from ietf-01 to ietf-02:

- Removed the requirement of periodic refreshing (#126).
- The new "Observe" Option replaces the "Lifetime" Option.
- Introduced a new mechanism to detect message reordering.

- Changed 2.00 (OK) notifications to 2.05 (Content) notifications.

Changes from ietf-00 to ietf-01:

- Changed terminology from "subscriptions" to "observation relationships" (#33).
- Changed the name of the option to "Lifetime".
- Clarified establishment of observation relationships.
- Clarified that an observation is only identified by the URI of the observed resource and the identity of the client (#66).
- Clarified rules for establishing observation relationships (#68).
- Clarified conditions under which an observation relationship is terminated.
- Added explanation on how clients can terminate an observation relationship before the lifetime ends (#34).
- Clarified that the overriding objective for notifications is eventual consistency of the actual and the observed state (#67).
- Specified how a server needs to deal with clients not acknowledging confirmable messages carrying notifications (#69).
- Added a mechanism to detect message reordering (#35).
- Added an explanation of how notifications can be cached, supporting both the freshness and the validation model (#39, #64).
- Clarified that non-GET requests do not affect observation relationships, and that GET requests without "Lifetime" Option affecting relationships is by design (#65).
- Described interaction with blockwise transfers (#36).
- Added Resource Discovery section (#99).
- Added IANA Considerations.
- Added Security Considerations (#40).
- Added examples (#38).

Author's Address

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

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

core
Internet-Draft
Intended status: Standards Track
Expires: November 20, 2011

K. Li
B. Greevenbosch
Huawei Technologies

May 19, 2011

CoAP Option Extension : Timeout
draft-li-core-coap-request-timeout-option-00

Abstract

CoAP is a RESTful application protocol for constrained nodes and networks. This specification provides a simple extension for CoAP, to inform a CoAP server of the maximum time that a CoAP client will wait a response to its request. A CoAP server can use this header to ensure that a timely response is generated.

Note

Discussion and suggestions for improvement are requested, and should be sent to 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 November 20, 2011.

Copyright Notice

Table of Contents

1. Introduction

This specification adds a new option Request-Timeout to CoAP. The main purpose is for the client to indicate the maximum time that a CoAP client will wait for a response to its request.

1.1. Justification

It is useful for the client to indicate that the response is required to be returned within a certain amount of time. For example, the client could require a response within 2 seconds. This applies to both a Piggy-backed Response and a Separate Response. With this indication of the response timeout, the client knows how long it should wait for the response, and it needs to keep the state of the request only for the indicated time. After this period, the request will be given up. In this way, the transmission resource can be saved to avoid the retransmission of requests. Also it can avoid that the server wastes resources by sending a response which already exceeds the set timeout of the client.

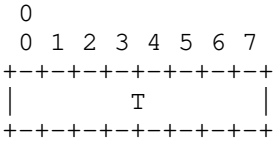
1.2. Terminology

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. Respuest-Timeout Option Extension

2.1. Request-Timeout Option Definition

Type	C/E	Name	Data type	Length	Default
20	E	Request-Timeout	uint	0-1 B	



Request-Timeout = 2^T milliseconds

The value of the Request-Timeout option is 2^T milliseconds. [TBD:

keep the spec open on whether the base is milliseconds or 1024s of a second -- the clocks are not going to be that precise anyway, and this might help some implementations that count in 1024s ("mibiseconds") or in whole seconds.]

2.2. Using the Request-Timeout Option

This option is used to indicate the maximum time that a client is prepared to wait for a response.

The client adds the Request-Timeout option to any request for which it is prepared to wait for a response. The client sets the option to the maximum time that it is prepared to wait.

The server interprets this option as the time between receipt of the complete request and the time that it generates and begins sending the response. The client will observe a longer time interval between request and response, as network transit and processing by proxies add delays. If timing is critical, the client SHOULD consider the possible delays and choose the value for the option accordingly.

The server MAY apply a lower value to the timeout based on local policy. A server MAY choose to take longer to produce a response, at the risk that the client is no longer able to use the response.

In case that the CoAP message is transmitted through a proxy, the Proxy MAY reduce the value of a Request-Timeout option based on a local policy. A Proxy MAY add a Request-Timeout option if none is present. The value in the Request-Timeout option MUST NOT be increased or removed.

If the client does not receive a response within the indicated response time, the client MAY consider the request as failed.

If the server can't provide response within the required time, a 5.XX (Can't provide the data in time) [TBD] MUST be returned. Note that the client cannot rely on getting the response code, because the server might have failed in the meantime.

This option is not used in a response.

This option is "elective". It MUST NOT occur more than once.

3. Example

This section gives a short example with a message flow that illustrates the use of the Request-Timeout option in a GET request.

The first example (Figure 1) shows that the client wants to get a response within 2048 milliseconds.

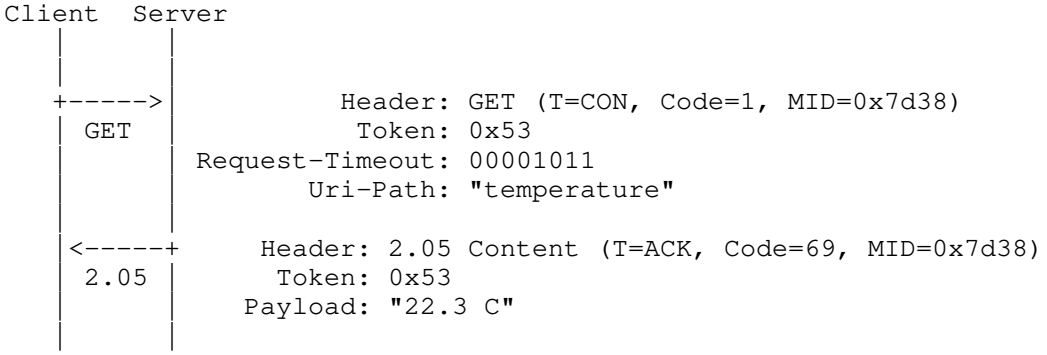


Figure 1: Request-Timeout Option in a request

Figure 1: Request-Timeout Option in a request

4. Security Considerations

This presents no security considerations beyond those in section 10 of the base CoAP specification [I-D.ietf-core-coap].

5. IANA Considerations

The IANA is requested to add the following Option Number entry.

Number	Name	Reference
20	Request-Timeout	Section 2

6. Acknowledgements

The authors of this draft would like to thank the participants of the email discussion on this issue. Thanks to Carsten Bormann, Peter Bigot, Barry Leiba, Linyi Tian for the reviews and discussions.

7. Normative References

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-03 (work in progress), May 2011.

[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-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-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

Kepeng Li
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P. R. China

Phone: +86-755-28974289
Email: likepeng@huawei.com

Bert Greevenbosch
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P. R. China

Phone: +86-755-28978088
Email: bgreeven@huawei.com

core
Internet-Draft
Intended status: Standards Track
Expires: April 23, 2012

K. Li
L. Tian
B. Leiba
Huawei Technologies
October 21, 2011

CoAP Option Extension : Size
draft-li-core-coap-size-option-02

Abstract

This document defines an extension to the Constrained Application Protocol (CoAP) to add a new option Size, which is used to indicate the resource size in a PUT/POST request or in a GET response.

Note

Discussion and suggestions for improvement are requested, and should be sent to 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 23, 2012.

Table of Contents

1. Introduction

This specification adds a new option Size to the Constrained Application Protocol (CoAP). The main purpose is to indicate the resource size in a PUT/POST request, or in a GET response.

1.1. Justification

If the requester wants to retrieve large resource data using a GET request, it is better to know in advance the size of the resource data. Currently in the Link Format [I-D.ietf-core-link-format] specification, the maximum size estimate attribute "sz" is defined to give an indication of the estimated maximum size of the resource data. By using this, the requester is able to know whether it is capable to accept the resource data. However it is not possible for the requester to know exactly how many blocks will be transmitted, therefore, concurrent GET can't be supported.

Also in a POST/PUT request (for example, a firmware update), it is not possible for the recipient to know in advance what is the size of the data to be transmitted. According to the current CoAP [I-D.ietf-core-coap] specification, when transmitting large data, the recipient will return an error code 4.13 (Request Entity Too Large) to the requester when the data size is too big to be accepted by the recipient. In this case the whole transmission has failed, and the previous received data will be useless. This is a waste of transmission resources.

This document adds the new Size Option to provide the capability to indicate the accurate size in a GET response or in a POST/PUT request.

By using the Size Option in a GET response, the CoAP Server can let the requester know the actual size of the resource in advance. This is especially useful for large resources, and can facilitate the requester to allocate enough buffer space before transmission. Also, using the block size, the requester can calculate the total number of blocks and can use concurrent GET requests to retrieve resource data using the Block Option. Finally, the recipient can check the resource size after the data transmission has been completed.

By using the Size Option in a PUT/POST request, the requester can indicate the resource size in the first Block Option message, to let the recipient know the resource data size in advance. If the recipient is not able to receive the data with the indicated size, the recipient can tell the requester in a response code, avoiding the cost of the actual data transmission.

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. Size Option Extension

2.1. Size Option Definition

Type	C/E	Name	Data type	Length	Default
18	E	Size	uint	0-4 B	

2.2. Using the Size Option

The Size Option is used to indicate the size of the resource data measured in bytes.

The GET request including Size=0 is treated as a request to get the size of the resource representation (but not the resource payload).

The GET request including an empty Size option is treated as a request to get the size of the resource representation with the resource payload.

The Size option MUST be included in the GET response, if the Size option is present in the request.

Also it SHOULD be used in a POST/PUT request in the first Block Option message.

The Size option SHOULD be included for resources larger than a single PDU, if the Size information is available. And it MAY be included for resources smaller than a single MTU.

In the absence of the option, the size of the resource data is calculated after the data has been transmitted to the recipient, either from the CoAP payload length or based on number of blocks and block size.

If the Size option is specified it SHOULD be accurate at that time, and SHOULD NOT be an estimate.

But due to the dynamic change of the resource data, the Size may not be accurate. If the value of Size option is not the same as the actual transmitted data, the recipient MUST take the size of the actual transmitted data as accurate, and ignore the Size option. In case that the recipient gets all the data but it is still smaller than the announced Size, the recipient SHOULD stop the transmission. If the recipient finds out the transmitted data reaches the Size limit, and there's more data left, the recipient SHOULD continue to transmit the remaining data.

This option is "Elective". It MUST NOT occur more than once.

3. Interaction with Block option

3.1. Usage in POST/PUT Request

In a PUT/POST request for large resource data, the requester SHOULD use the Size option to indicate the size of the resource. If the recipient is not capable to receive the data with the indicated size, the recipient MUST return a 4.13 (Request Entity Too Large) response code to the requester, and the data transmission is avoided, so that the cost of the actual data transmission is saved.

3.2. Usage in GET Response

In a GET response for large resource data, the CoAP Server SHOULD use the Size option to indicate the resource size and return the first block data. The requester can calculate the number of blocks to be transferred based on the block size and the resource size, and use concurrent GET requests to retrieve resource data. Also, when the client determines it cannot process data of this Size, it MAY choose to abort and not to send subsequent GETs.

4. How to merge into Block draft

This section introduces how to merge the Size option draft into Block draft with the minimum functionalities.

4.1. The Size option

This section will work as section 2.3 in Block draft.

Type	C/E	Name	Data type	Length	Default
18	E	Size	uint	1-4 B	

4.2. Using the Size option

This section will work as section 2.4 in Block draft.

The Size Option is used to indicate the size of the resource data measured in bytes.

The Size option SHOULD be used in a POST/PUT request in the first Block Option message. If the recipient is not capable to receive the data with the indicated size, the recipient MUST return a 4.13 (Request Entity Too Large) response code to the requester, and the data transmission is avoided, so that the cost of the actual data transmission is saved.

For a GET request, if it includes an empty Size option, the Size option MUST be included in the response. If the GET request includes a Block option, the Size option SHOULD be included in the first Block response. In other cases the GET response MAY contain a Size option.

If the Size option is specified, it SHOULD be accurate at that time, and SHOULD NOT be an estimate.

The option is "Elective". It MUST NOT occur more than once.

4.3. Example

Example as indicated as Figure 2 in this draft can be added in section 3 in the Block draft.

5. Examples

This section gives a number of short examples with message flows to illustrate the use of Size option in a GET response, or in a PUT/POST request.

The first example (Figure 1) shows that the requester does not know the resource data size, and sends the GET request, the recipient can send back the resource size using the Size option and the first block. In the subsequent GET request, the requester can calculate the number of blocks and use concurrent GET requests to retrieve the resource data.

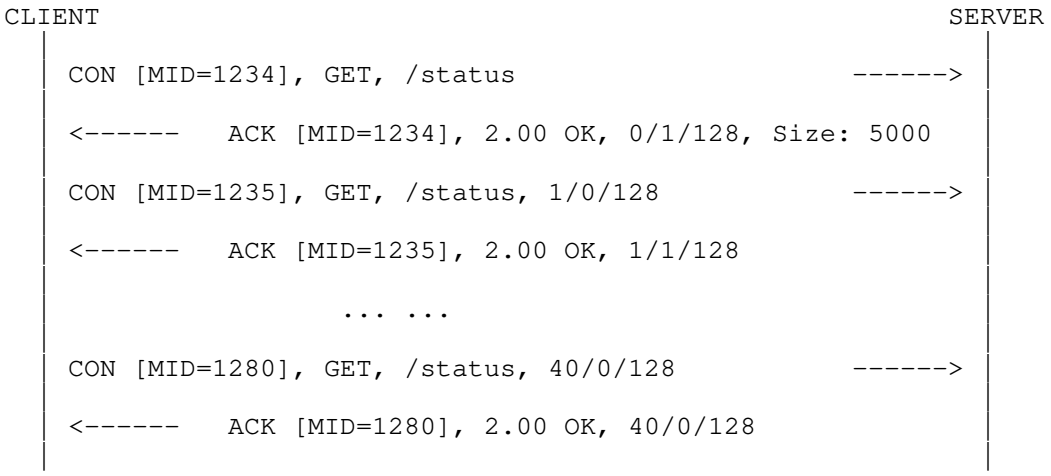


Figure 1: Size Option in a GET response

The second example (Figure 2) shows the requester sending a PUT request with the Size option to indicate the resource data size, and since the recipient determines that the resource data is too large to be accepted, it sends back a 4.13 (Request Entity Too Large) response code.

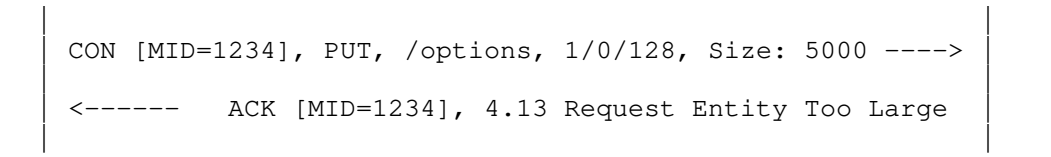


Figure 2: Size Option in a PUT request

6. Security Considerations

As the size option is used to determine whether or not the reciepient will accept the data, lying about it can cause the recipient to make a wrong decision. For example, an attacker might reduce the reported size such that the recipient will accept, even when it cannot process the complete data.

Related is another attack, where the attacker changes the reported size to a higher value, leading to the recipient rejecting even when it has the capability to receive.

The latter attack is similar to an attack where the attacker blocks

the packets altogether; although it is more efficient since the attacker only needs to modify one message. The former attack needs serious consideration at implementation level, especially concerning possible buffer overflows that might lead to data leaking into the code.

7. IANA Considerations

The IANA is requested to add the following Option Number entry.

Number	Name	Reference
18	Size	Section 2

8. Acknowledgements

The authors of this draft would like to thank the participants of the email discussion on this issue. Thanks to Bert Greevenbosch, Charles Palmer and Carsten Bormann for the detailed reviews and suggestions.

9. Normative References

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-04 (work in progress), July 2011.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-07 (work in progress), July 2011.

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

Authors' Addresses

Kepeng Li
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P. R. China

Phone: +86-755-28974289
Email: likepeng@huawei.com

Linyi Tian
Huawei Technologies
Huawei Base, Bantian, Longgang District
Shenzhen, Guangdong 518129
P. R. China

Phone: +86-755-28978078
Email: tianlinyi@huawei.com

Barry Leiba
Huawei Technologies

Phone: +1 646 827 0648
Email: barryleiba@computer.org
URI: http://internetmessagingtechnology.org/

This Internet-Draft, draft-lynn-core-discovery-mapping-01.txt, has expired, and has been deleted from the Internet-Drafts directory. An Internet-Draft expires 185 days from the date that it is posted unless it is replaced by an updated version, or the Secretariat has been notified that the document is under official review by the IESG or has been passed to the RFC Editor for review and/or publication as an RFC. This Internet-Draft was not published as an RFC.

Internet-Drafts are not archival documents, and copies of Internet-Drafts that have been deleted from the directory are not available. The Secretariat does not have any information regarding the future plans of the authors or working group, if applicable, with respect to this deleted Internet-Draft. For more information, or to request a copy of the document, please contact the authors directly.

Draft Authors:
Kerry Lynn<kerlyn@ieee.org>
Zach Shelby<zach@sensinode.com>

This Internet-Draft, draft-ohba-core-eap-based-bootstrapping-00.txt, has expired, and has been deleted from the Internet-Drafts directory. An Internet-Draft expires 185 days from the date that it is posted unless it is replaced by an updated version, or the Secretariat has been notified that the document is under official review by the IESG or has been passed to the RFC Editor for review and/or publication as an RFC. This Internet-Draft was not published as an RFC.

Internet-Drafts are not archival documents, and copies of Internet-Drafts that have been deleted from the directory are not available. The Secretariat does not have any information regarding the future plans of the authors or working group, if applicable, with respect to this deleted Internet-Draft. For more information, or to request a copy of the document, please contact the authors directly.

Draft Authors:
Subir Das<subir@research.telcordia.com>
Yoshihiro Ohba<yoshihiro.ohba@toshiba.co.jp>

CoRE
Internet-Draft
Intended status: Informational
Expires: April 14, 2012

A. Rahman, Ed.
InterDigital Communications, LLC
E. Dijk, Ed.
Philips Research
October 12, 2011

Group Communication for CoAP
draft-rahman-core-groupcomm-07

Abstract

This is a working document intended to trigger discussion and develop draft text for the CoAP protocol specification in the area of group communication. Engineering tradeoffs become more challenging in constrained environments, therefore group communication is considered within the context of adjacent topics that may impact or be impacted by design choices in the subject area. A solution based on IP multicast is proposed.

Table of Contents

1. Conventions and 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 following are definitions of specific terminology used in this draft.

Group Communication: A source node sends a message to more than one destination node, where all destinations are identified to belong to a specific group. The set of source nodes and destination nodes may consist of an arbitrary mix of constrained and non-constrained nodes. Sending methods may include serial unicast, multicast, or hybrid unicast-to-multicast solutions.

Multicast: Sending a message to multiple receiving nodes simultaneously. Typically, this is done as part of a group communication process. There are various options to implement multicast including layer 2 (Media Access Control) or layer 3 (IP) mechanisms.

IP Multicast: A specific multicast solution 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].

Low power and Lossy Network (LLN): LLNs are made up of constrained devices. These devices may be interconnected by a variety of links, such as IEEE 802.15.4, Bluetooth, WiFi, wired or low-power powerline communication links.

2. Introduction

2.1. Background

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 CoAP are documented in [I-D.shelby-core-coap-req].

Constrained devices can be large in number, but highly correlated to each other. For example, all the light switches in a building may belong to one group and all the thermostats belong to another group. All the smart meters in the same region can belong to a group as well. Groups may be composed by function; for example, the group "all lights in building one" may consist of the groups "all lights on

floor one of building one", "all lights on floor two of building one", etc. Groups may also be configured or dynamically formed.

In this draft, we focus and expand discussions on the requirements pertaining to CoAP "group communication" and "multicast" support including:

> 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:

> "CoAP supports sending messages to multicast destination addresses. Such multicast messages MUST be Non-Confirmable. 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).

2.2. Problem Statement and Scope

In this draft, we expand the scope from unreliable IP multicast in the current CoAP requirement to group communication, using either (reliable or unreliable) multicast or unicast or combinations thereof. We assume that all, or a substantial part of, devices participating in group communication are constrained devices (e.g. such as Low Power and Lossy Network (LLN) devices).

Machine-to-Machine (M2M) networks may contain groups of nodes that are highly correlated (e.g. by type or location). For example, all smart meters in a region may belong to one group, and all light switches in a building control system belong to another. Group communication mechanisms can improve efficiency and latency of communication and reduce bandwidth requirements for a given application.

In the following sections, we address the issues related to group communication in detail, with requirements, proposed solutions and analysis of their impact to the CoAP protocol and implementations.

3. Potential Solutions

3.1. Overview

The classic concept of group communications is that of a single source distributing content to multiple recipients that are all part of a group, as shown in the example sequence diagram in Figure 1. Also shown there is the pre-requisite step of forming the group before content can be distributed to it.

Group communication solutions have evolved from "bottom" to "top", i.e., from the network layer (IP multicast) to application layer group communication, also referred to as application layer multicast. A study published in 2005 [STUDY1] 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 [STUDY1] using metrics such as link stress and level of host complexity [STUDY2]. The results show for a realistic internet topology that IP Multicast is most resource-efficient, with the only downside being that it requires most effort to deploy in the infrastructure.

The approach adopted in this section is to begin with group communication requirements. This is followed by the solutions of IP multicast, an overlay multicast solution, and application layer group communication. Finally additional topics are covered such as group management and CoAP/HTTP proxies in group communication.

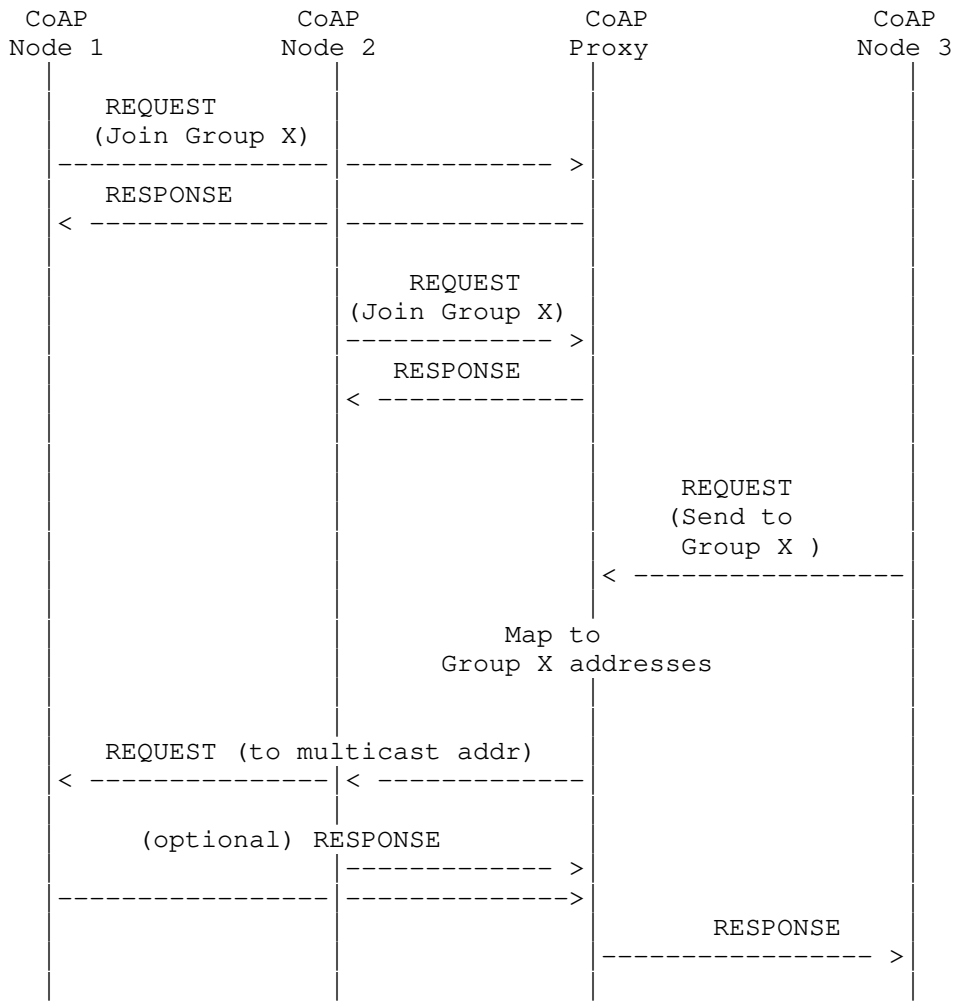


Figure 1: CoAP Group Communications

3.2. Requirements

Requirements that a group communication solution in CoRE should fulfill can be found in existing documents [RFC 5867] [draft-ietf-6lowpan-routing-requirements] [I-D.vanderstok-core-bc] [I-D.shelby-core-coap-req]. Below, a set of high-level requirements is listed that a group communication solution in CoRE should ideally fulfill. More precise requirements may depend on the chosen application (area).

A CoRE group communication solution should (ideally) offer:

REQ 1: Optional Reliability: unreliable group communication, with preferably reliable group communication as an option.

REQ 2: Efficiency: delivers messages more efficiently than a "serial unicast only" solution. Also, it should provide a right balance between group data traffic and control overhead.

REQ 3: Low latency: deliver a message (preferably) as fast as possible.

REQ 4: Synchrony: allows near-simultaneous modification of a resource on all devices in a group, providing to users a perceived effect of synchrony or simultaneity. It can be expressed as a time span D such that message m is delivered to all destinations in a time interval [t,t+D] for arbitrary t.

REQ 5: Ordering: message ordering in the reliable group communication mode.

REQ 6: Security: see Section 5 for security requirements for group communication.

REQ 7: Flexibility: support for one or many source(s), for dense and sparse networks, for high or low listener density, one or many group(s), and multi-group membership.

REQ 8: Robust group management: includes functionality to join groups, leave groups, view group membership, and persistent group membership in failing node or sleeping node situations.

REQ 9: Network layer independence: a solution should be specified independent from specific unicast and/or IP multicast routing protocols. It should support different routing protocols and implementations thereof.

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.

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.

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).

REQ 13: CoAP Proxying support: a CoAP proxy can handle distribution of a message to a group on behalf of a (constrained) CoAP client.
REQ 14: Suitable for operation on LLNs with constrained nodes.

3.3. IP Multicast

IP Multicast protocols have been evolving for decades, resulting in proposed standards such as Protocol Independent Multicast - Sparse Mode (PIM-SM) [RFC4601]. Yet, due to various technical and marketing reasons, IP Multicast is not widely deployed on the general Internet. However, IP Multicast is popular in specific deployments such as in enterprise networks (e.g. for video conferencing or general IP multicast PC applications within a single LAN broadcast domain) and carrier IPTV deployments. Therefore, the packet economy and minimal host complexity of IP multicast make it worth investigating for group communication in constrained environments.

3.3.1. Multicast Listener Discovery (MLD) and Multicast Router Discovery (MRD)

The Multicast Listener Discovery (MLD) protocol [RFC3810] (or its IPv4 pendant IGMP) is today the method of choice used by an (IP multicast enabled) IPv6 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. It 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 a 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 switch 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 an MLD Router) if no MLD router is present.

The Multicast Router Discovery (MRD) protocol [RFC4286] defines a way to discover multicast routers, for the purpose of using this information by IGMP/MLD snooping devices. However, it appears that this protocol is not as commonly implemented in existing products as MLD is.

3.3.2. Group URIs and Multicast Addresses

An approach to map group authorities onto IP multicast addresses using DNS was proposed in [I-D.vanderstok-core-bc]. Examples of group URI naming (and scoping) for a building control application are shown below. Group URIs MUST follow the approach of the URI structure defined in [RFC3986].

URI authority	Targeted group
all.bldg6...	"all nodes in building 6"
all.west.bldg6...	"all nodes in west wing, building 6"
all.floor1.west.bldg6...	"all nodes in floor 1, west wing, etc."
all.bu036.floor1.west.bldg6...	"all nodes in office bu036, floor1, etc."

The authority portion of the URI is used to identify a node (or group) and the resulting DNS name is bound to a unicast or multicast IP address. Each example group URI shown above can be mapped to a unique multicast IP address. This may be an address allocated according to [RFC3956], [RFC3306] or [RFC3307].

3.3.3. Group Discovery

CoAP defines a resource discovery capability but, in the absence of a standardized group communication infrastructure, it is limited to link-local scope IP multicast; examples may be found in [I-D.ietf-core-link-format]. A service discovery capability is required to extend discovery to other subnets and scale beyond a certain point, as originally proposed in [I-D.vanderstok-core-bc].

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 or alternatively the name _coap._upd.<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 subtypes 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>.

3.3.4. Group Resource Manipulation

At least two forms of group resource manipulation must be supported. The first is push (multicast PUT or MPUT for short) as e.g. "turn off all the lights simultaneously". Logically, this is similar to publishing a value to multiple subscribers. The second operation is pull (multicast GET or MGET), which is essential for discovery during commissioning and can be illustrated by the example "return all the resources on all CoAP servers advertized by their .well-known/core URI". MGET to an "all-nodes" or "all-CoAP-nodes" multicast IP address should perhaps be limited in scope to link-local multicast for scaling [TBD: and possibly for security reasons, e.g. DoS attacks].

Conceptually, the result of a multicast GET or PUT should be the same as if the client had unicast them serially (that is, a set of {URI, representation} tuples). Practically, there are major benefits to avoiding serial unicast in favor of a multicast CoAP GET/PUT solution:

- packet economy on constrained networks
- M2M resource discovery (solves the "chicken-and-egg" problem)
- apparent simultaneity of events (e.g. in lighting applications)
- average lower latency per event (e.g. in lighting applications)

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 Section 3.3.3 and [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 requests.

A second solution is to impose the following restrictions, e.g. for groups not found using, or advertized in, DNS-SD:

- All CoAP multicast requests MUST be sent to the well-known CoAP port.
- All CoAP multicast requests SHOULD operate on /.well-known/core URIs

One question is whether the application (or middleboxes) need to be aware that a request is intended for a group. A separate scheme as proposed by [ID.goland-http-udp] might be useful (e.g. "corem" vs. "core"). To the extent that group membership might be implemented as a series of IP multicast, serial unicast, or some combination, having a distinct scheme for group operations might be a useful signal for a proxy receiving the request to look up the group membership and replicate serial unicasts as well as send multicast packets.

3.3.5. IP Multicast Transmission Methods

3.3.5.1. 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.

3.3.5.2. 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.

3.3.5.3. 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]:

- 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.
- Integrity - destination receives m at most once from sender and only if sender sent m to a group including destination.
- Agreement - If a correct destination of g receives m, then all correct destinations of g receive m.
- 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

- 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.
- Route redundancy
- Source node transmits multiple times (destinations do not respond)

3.3.6. Congestion Control

CoAP requests may be multicast, resulting a multitude of replies from different nodes, potentially causing congestion. [I-D.eggert-core-congestion-control] suggests to conservatively control sending multicast requests.

CoAP already addresses the congestion problem to some extent by requiring all multicast CoAP requests to be Non-Confirmable. However, as responses to multicast requests (both MGET or MPUT) are required in CoAP, using CoAP multicast still may lead to congestion issues.

Various means can be implemented to prevent congestion.

[TBD: if an MGET or MPUT request leads to the sending of a CoAP response by servers, the servers should enforce a random delay within TIMEOUT before sending their responses. More investigation required.]

Currently in the CoAP protocol, a MAX_RETRANSMIT value set by default to 4 is used for retransmission of Confirmable messages. Since CoAP multicast messages are Non-Confirmable, no retransmissions will occur in CoAP, making the effective retransmission value 0.

3.4. 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 (Section 3.3.1) 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.

3.5. 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:
- discover groups;
- query group properties (e.g. related resource descriptions);
- create a group;
- remove a group;
- add a group member;
- remove a group member;
- enumerate group members;
- 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 2:

Type	C/E	Name	Data type	Length	Default
x	E	Group Join	String	1-270 B	""
x+2	E	Group Leave	String	1-270 B	""

Figure 2: CoAP Header Options for Group Management

Figure 3 illustrates how a node can join or leave a group using the Header Options in a CoAP message:

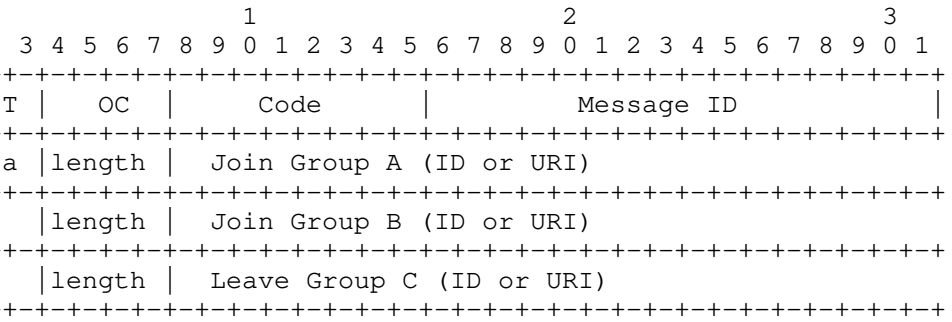


Figure 3: CoAP Message for Group Management

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 3, 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 advertized by DNS-SD. For example, to join group1 and leave group2:

coap://proxy1.bld2.example.com/groupmgt?j=group1&l=group2

3.6. CoAP Multicast and HTTP Unicast Interworking

Within the constrained network, CoAP runs over UDP for which IP multicast is supported. In a non-constrained network (i.e. general Internet), HTTP over TCP is used for which IP multicast is not supported. Therefore a CoAP/HTTP Proxy node that supports group communication needs to have functionalities to support interworking of unicast and multicast. One possible way of operation of the Proxy is illustrated in Figure 4. Note that this topic is covered in more detail in [I-D.castellani-core-http-mapping].

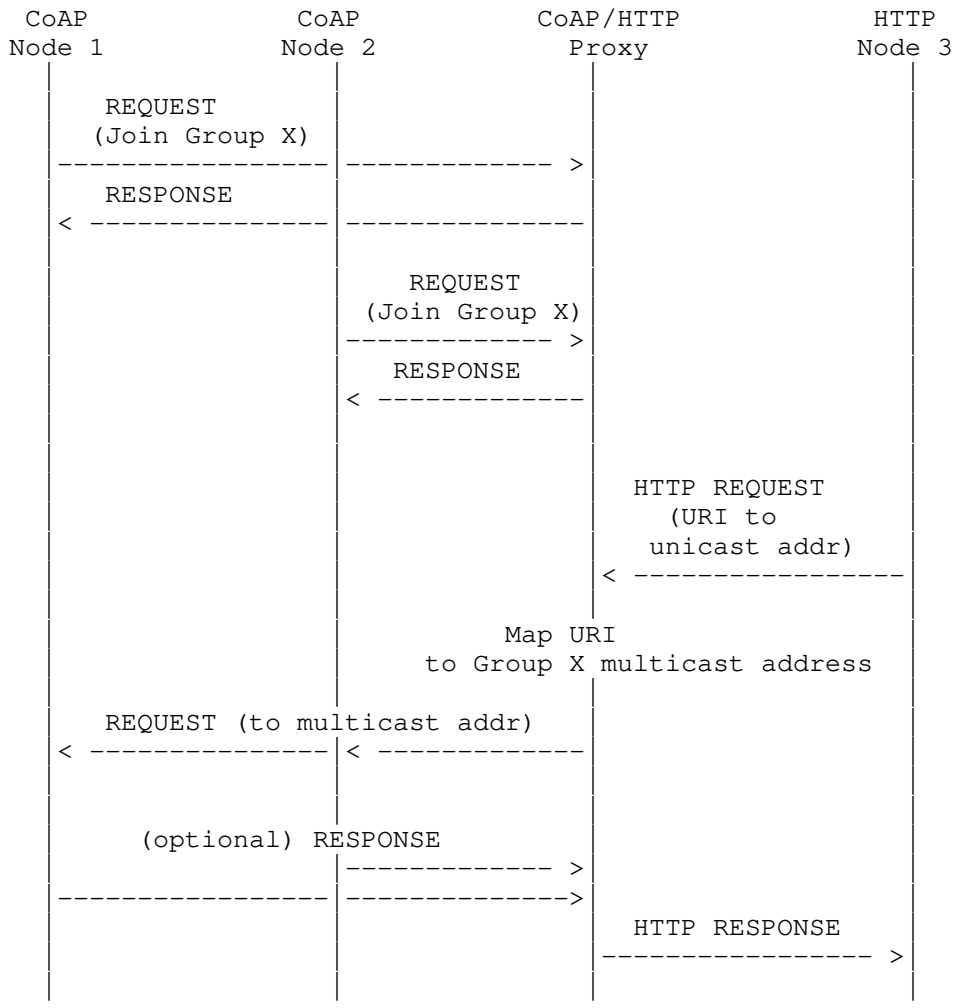


Figure 4: CoAP Multicast and HTTP Unicast Interworking

Note that Figure 4 illustrates the case of IP multicast as the underlying group communications mechanism. However the overlay multicast group communication (Section 3.4) or CoAP application group communication (Section 3.5) can be used as the underlying mechanism and the principles of the figure would still apply (i.e. CoAP proxy needs to do interworking between HTTP unicast and CoAP multicast).

A key point in Figure 4 is that the incoming HTTP Request (from node 3) will carry a URI (with the HTTP scheme) that resolves in the general Internet to the proxy node. At the proxy node, the URI will

then possibly be mapped (as detailed in [I-D.castellani-core-http-mapping]) and again resolved (with the CoAP scheme) to an IP multicast destination. This may be accomplished, for example, by using DNS-SD (Section 3.3.3). The proxy node will then IP multicast the CoAP Request (corresponding to the received HTTP Request) to the appropriate nodes (i.e. nodes 1 and 2).

In terms of the HTTP Response, Figure 4 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-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 could be carried in the HTTP Request Line (as defined in [I-D.ietf-core-coap] Section 8) in a HTTP request sent to the IP address of the Proxy.

3.7. CoAP-Observe for Group Communication

The CoAP Observation extension [I-D.ietf-core-observe] can be directly used for group communication. A group then 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 in this case for notifications.

Group communication 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.

4. Recommended Solution

4.1. Overview

We recommend that IP multicast as outlined in Section 3.3 be adopted as the base solution for CoAP Group Communication. This approach re-uses the IP multicast suite of protocols and can operate on both constrained and non-constrained network segments. The group communication can hence work regardless of the underlying networking technology. 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.

4.2. An Example Protocol Flow

We first present an example use case to illustrate the overall steps in an IP Multicast based CoAP Group Communication solution. We assume the following network configuration for this example (see Figure 5):

1) 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 6LoWPAN subnets.

2) Light-1 and the Light Switch are connected to a router (Rtr-1) which is also a CoAP Proxy and a 6LoWPAN Border Router (6LBR).

3) Light-2 and the Light-3 are connected to another router (Rtr-2) which is also a CoAP Proxy and a 6LBR.

4) The routers are connected to a an IPv6 network backbone which is also multicast enabled. In the general case, this means the network backbone and 6LBRs support a PIM based multicast routing protocol, and MLD for forming groups. In a limited case, if the network backbone is one link, then the routers only have to support MLD-snooping for the example use case to work.

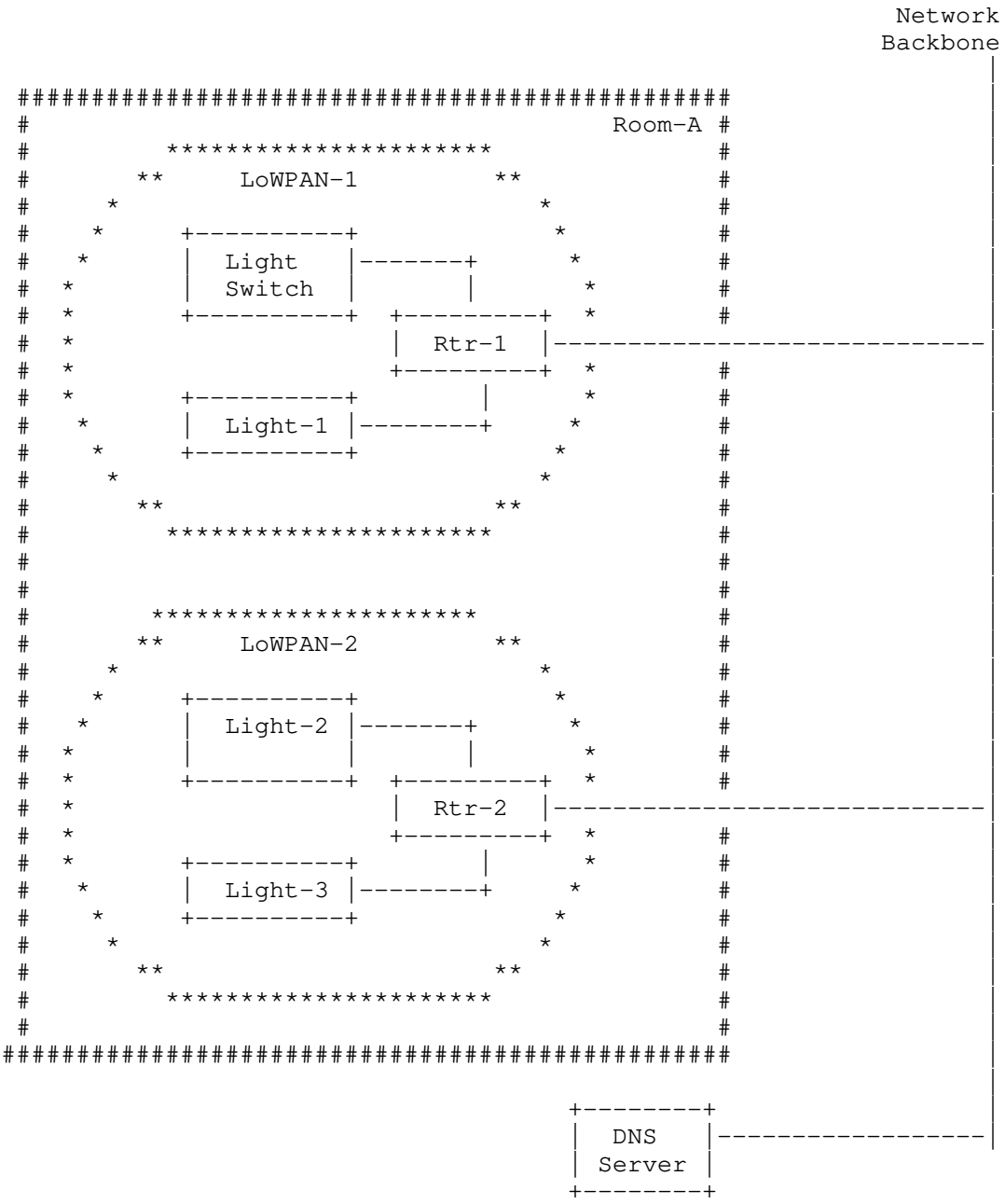
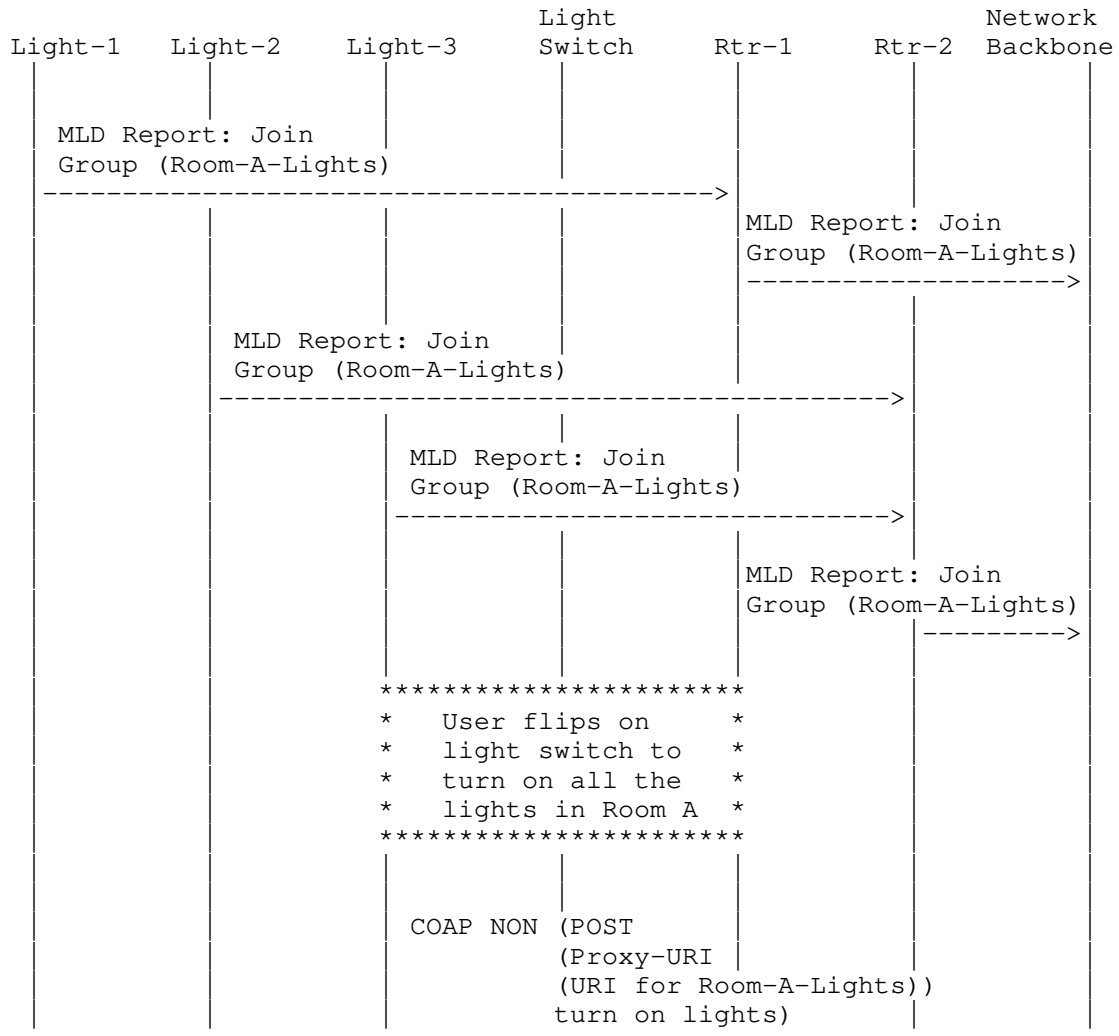


Figure 5: Network Topology of a Large Room (Room-A)

The corresponding protocol flow for an IP Multicast based CoAP Group Communication solution for the network shown in Figure 5 is shown in Figure 6. We assume the following steps occur before the illustrated flow:

1) Startup phase: 6LoWPANs are formed. IPv6 addresses assigned to all devices. The CoAP network is formed.

2) Commissioning phase (by applications): The IP multicast address of the group (Room-A-Lights) has been set in all the Lights. The URI of the group (Room-A-Lights) has been set in the Light Switch.



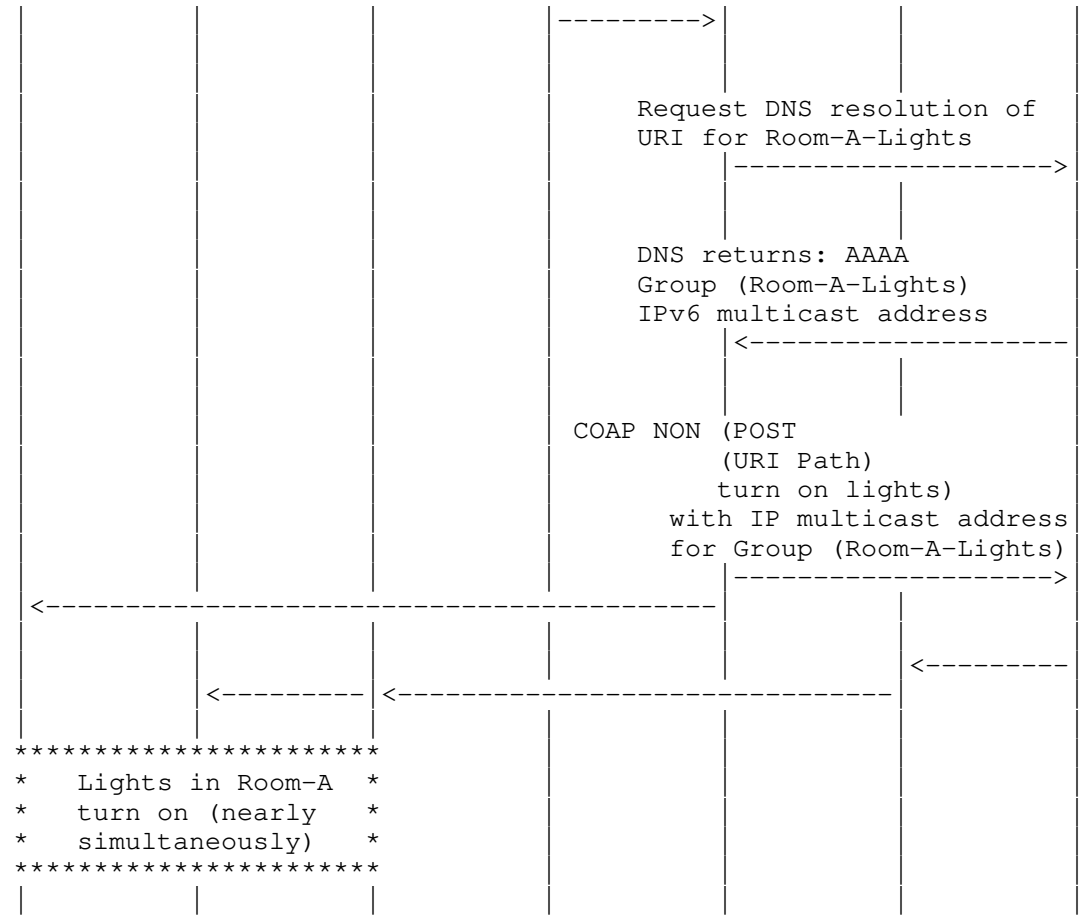


Figure 6: Turning on Lights in a Large Room (Room-A)

The indicated MLD Report messages are link-local multicast. In each LoWPAN, it is assumed that a multicast routing protocol in 6LRs will propagate the Join information over multiple hops to the 6LBR.

4.3. 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.)

4.3.1. 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.

4.3.1.1. 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 througout 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 Section 3.3.1). 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.

4.3.1.2. RPL Multicast Routing

The RPL routing protocol for LLNs provides support for routing to multicast IP destinations (Section 12 of [I-D.ietf-roll-rpl]). Like regular unicast destinations, multicast destinations are advertized 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.

4.3.1.3. 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 use link-local RPL routers for their IP connectivity. Note that the current RPL specification [I-D.ietf-roll-rpl] considers this case to be out of scope. However, it was suggested on the ROLL mailing list that RPL could potentially be run with non-RPL-aware hosts but that it is simply not specified yet. Such non-RPL hosts can't advertize their IP multicast groups of interest via RPL DAO messages as defined above. Therefore in that case MLD can be used for such advertizements (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 4.5.1.

4.3.1.4. 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.

4.3.1.5. 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 advertize 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 4.5.1 for more efficient substitutes for MLD targeted towards a LLN context.

4.3.2. 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.

4.3.2.1. 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 learnt 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 4.5.1.

4.3.2.2. 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.

4.3.2.3. 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 4.3.1.3.

4.3.2.4. 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 4.3.2.1.

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 4.3.2.1) of potentially overloading the LLN with unwanted backbone IP multicast traffic appears again.

A possible solution to this is (again) to let multicast listeners advertize their interest using MLD as described in Section 4.3.2.1 or to use an MLD alternative suitable for LLNs as described in Section 4.5.1. However, following this approach requires possibly an

extension to Trickle Multicast Forwarding: the protocol should ensure that MLD-advertized information is somehow communicated to the 6LBR, possibly over multiple hops. MLD itself supports link-local communication only.

4.3.2.5. 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.

4.3.3. 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.

4.3.4. 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?]

4.3.5. 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 interwork with regular MLD on the backbone network. Such MLD variants are further analyzed in Section 4.5.1.

4.4. HTTP/CoAP Interworking Aspects

The topic of HTTP unicast to CoAP multicast request proxying is treated in [I-D.castellani-core-http-mapping]. [TBD: only if needed more information will be added here in the future.]

4.5. Implementation Considerations

In this section various implementation aspects are considered such as required protocol implementations, additional functionality of the 6LBR and backbone network equipment.

4.5.1. MLD Implementation on LLNs

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 Snoopers, passively listen to MLD State Change Report messages and handle the learnt ("snooped") IP multicast destinations in the way defined by the multicast routing protocol they are running (e.g. for RPL, Routers advertize 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.

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 [I-D.ietf-6lowpan-nd] 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 unicast IP address, a host "registers" its interest in a multicast IP address.

Unlike ARO, multiple MLO can be used in the same ND packet. A registration period is also defined 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 full MLD.

[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]

4.5.2. 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.

4.5.3. 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-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 & deployed IP (Ethernet/WiFi/...) infrastructure, to be shared with other IP devices (PCs)
3. Newly designed & 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.]

5. Security Considerations

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.

Note that some of the requirements are marked optional. This means that, depending on the use case, these may be required or not. For this purpose each use case can be associated to a security profile as specified in [I-D.garcia-core-security]. The security profile prescribes what requirements should be taken into account for this profile. A mapping of these requirements to these profiles has not yet been done.

REQ1- 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.

REQ2- 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.

REQ3- 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.

REQ4- 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.

REQ5- 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.

REQ6- 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 [I-D.ietf-roll-rpl]. 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.

REQ7- 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.

6. IANA Considerations

This document makes no request of IANA.

7. Conclusions

Three solutions for enabling CoAP group communications have been discussed.

Unreliable IP multicast as outlined in Section 3.3 is recommended to be adopted as the base solution for CoAP Group Communication on LLNs. This approach requires no standards changes to the IP multicast suite of protocols and it provides interoperability with IP multicast group communication on unconstrained backbone networks.

The proposals for group communication described in this draft should be considered for incorporation into the overall CoAP protocol specification.

8. Acknowledgements

Thanks to Peter Bigot, Carsten Bormann, Anders Brandt, Angelo

Castellani, Guang Lu, Salvatore Loreto, Kerry Lynn, Dale Seed, Zach Shelby, Peter van der Stok, and Juan Carlos Zuniga for their helpful comments and discussions that have helped shape this document.

9. References

9.1. 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.

[RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, August 2002.

[RFC3307] Haberman, B., "Allocation Guidelines for IPv6 Multicast Addresses", RFC 3307, August 2002.

[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.

[RFC3956] Savola, P. and B. Haberman, "Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address", RFC 3956, November 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, April 2005.

[RFC4286] Haberman, B. and J. Martin, "Multicast Router Discovery", RFC 4286, December 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.

[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.

[RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, March 2010.

9.2. Informative References

[I-D.cheshire-dnsext-dns-sd]
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-10 (work in progress), February 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-hc]
Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams in Low Power and Lossy Networks (6LoWPAN)", draft-ietf-6lowpan-hc-15 (work in progress), February 2011.

[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-17 (work in progress), June 2011.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-07 (work in progress), July 2011.

[I-D.ietf-core-observe]
Hartke, K. and Z. Shelby, "Observing Resources in CoAP", draft-ietf-core-observe-02 (work in progress), March 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.vanderstok-core-bc]
Stok, P. and K. Lynn, "CoAP Utilization for Building Control", draft-vanderstok-core-bc-04 (work in progress), July 2011.

[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-01 (work in progress), July 2011.

[I-D.garcia-core-security]
Garcia-Morchon, O., Keoh, S., Kumar, S., Hummen, R., and R. Struik, "Security Considerations in the IP-based Internet of Things", draft-garcia-core-security-02 (work in progress), July 2011.

[I-D.ietf-roll-rpl]
Winter, T., Thubert, P., Brandt, A., Clausen, T., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., and J. Vasseur, "RPL: IPv6 Routing Protocol for Low power and Lossy Networks", draft-ietf-roll-rpl-19 (work in progress), March 2011.

[I-D.ietf-roll-trickle-mcast]
Hui, J. and R. Kelsey, "Multicast Forwarding Using Trickle", draft-ietf-roll-trickle-mcast-00 (work in progress), April 2011.

[ID.goland-http-udp]
Goland, Y., "Multicast and Unicast UDP HTTP Messages", 1999, <http://tools.ietf.org/html/draft-goland-http-udp-01>.

[STUDY1] 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>.

[STUDY2] Banerjee, B. and B. Bhattacharjee, "A Comparative Study of Application Layer Multicast Protocols", 2001, <http://wmedia.grnet.gr/P2PBackground/a-comparative-study-ofALM.pdf>.

Authors' Addresses

Akbar Rahman (editor)
InterDigital Communications, LLC

Email: Akbar.Rahman@InterDigital.com

Esko Dijk (editor)
Philips Research

Email: esko.dijk@philips.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.

Table of Contents

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 registrer, 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 periodially 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.

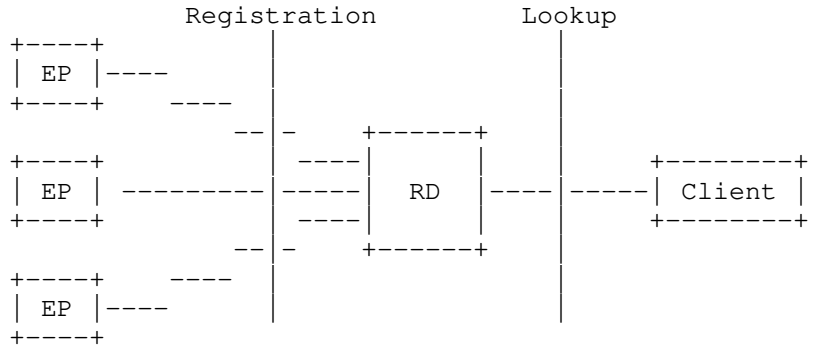


Figure 1: The resource directory architecture.

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, submiting 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 initate 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 commision 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

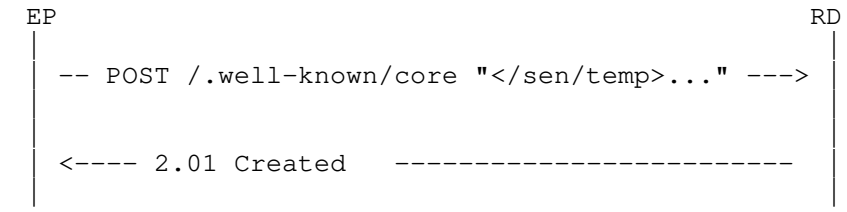
- empty, in which case the directory server is encouraged by this POST request to perform GET requests at the requesting server's default discovery URI.

or

- 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.



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:

- specific static configuration (e.g., anycast addresses), if any,
- the ABRO option of 6LoWPAN-ND [RFC6775],
- 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:

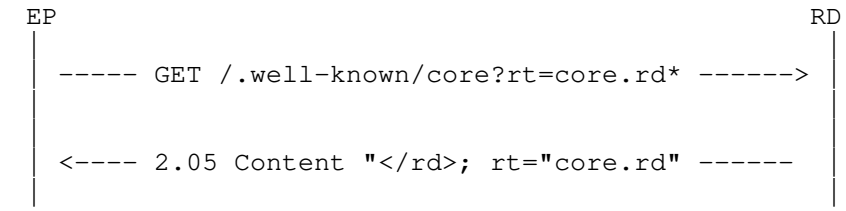
Success: 2.05 "Content" with an application/link-format payload containing a matching entry for the RD resource.

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.



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:

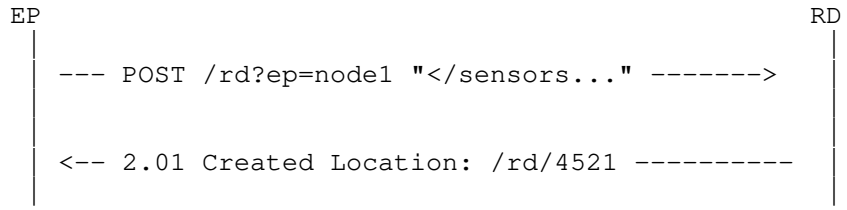
Success: 2.01 "Created". The Location header MUST be included with the new resource entry for the endpoint. This Location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration (update, delete).

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.



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 btyes. 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

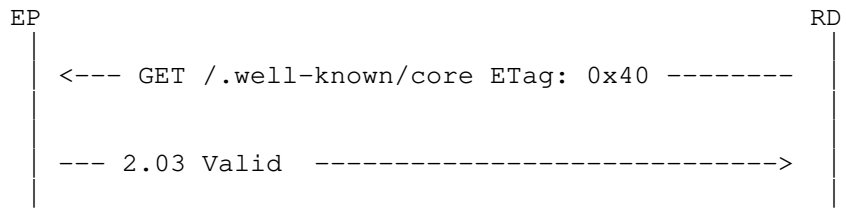
The following responses codes are defined for this interface:

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.



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.

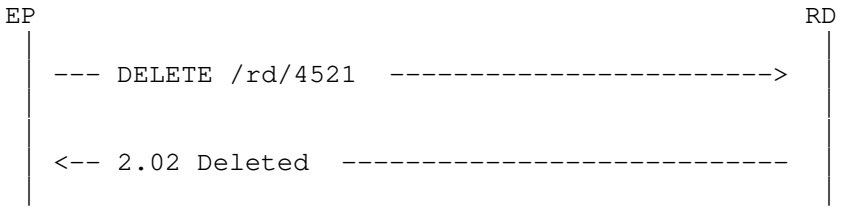
The following responses codes are defined for this interface:

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

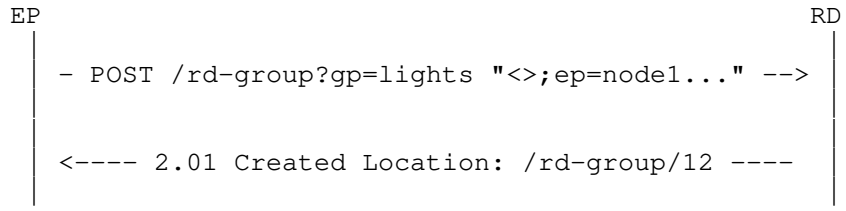
The following response codes are defined for this interface:

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.



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.

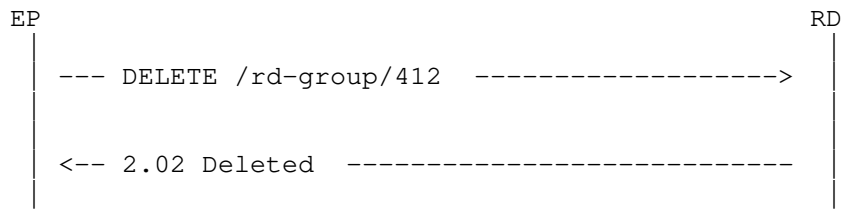
The following responses codes are defined for this interface:

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.



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.

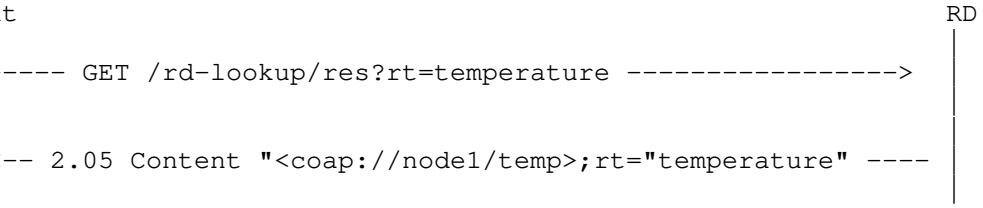
Failure: 4.04 "Not Found" in case no matching entry is found for a unicast request.

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.

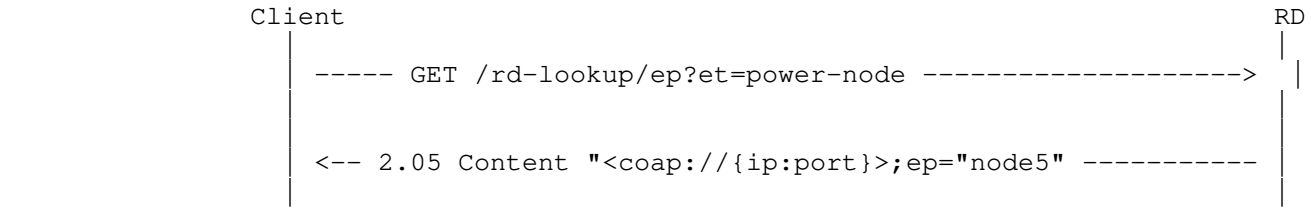
The following example shows a client performing a resource lookup:



Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content
<coap://{ip:port}/temp>

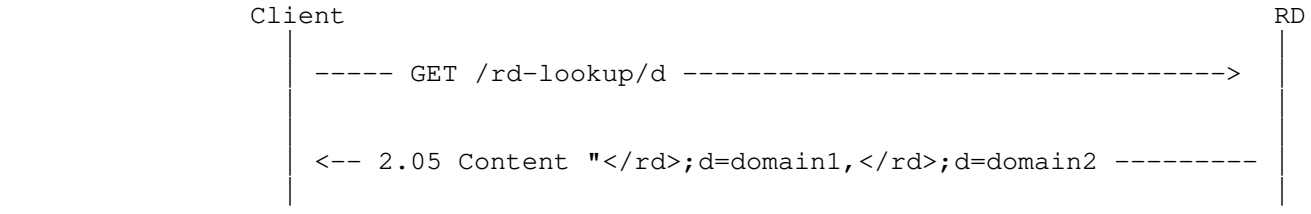
The following example shows a client performing an endpoint lookup:



Req: GET /rd-lookup/ep?et=power-node

Res: 2.05 Content
<coap://{ip:port}>;ep="node5",
<coap://{ip:port}>;ep="node7"

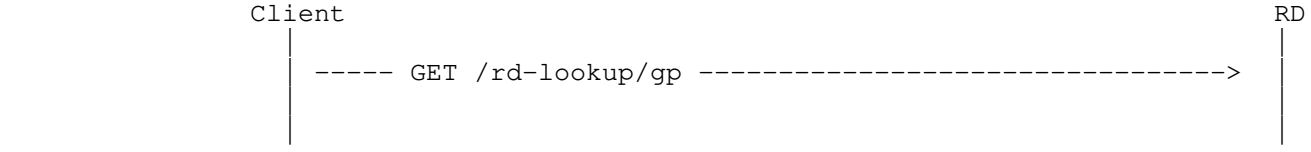
The following example shows a client performing a domain lookup:

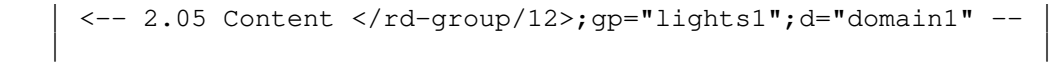


Req: GET /rd-lookup/d

Res: 2.05 Content
</rd>;d="domain1",
</rd>;d="domain2"

The following example shows a client performing a group lookup for all groups:

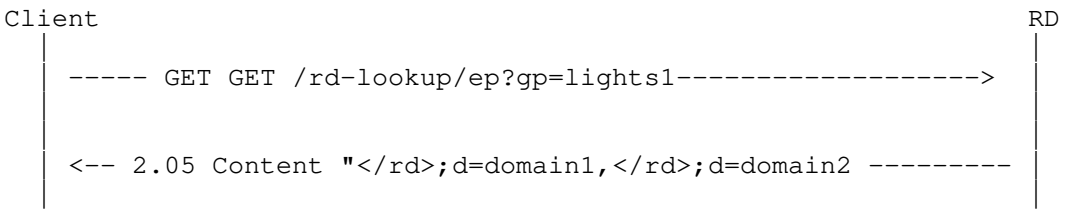




Req: GET /rd-lookup/gp

Res: 2.05 Content
</rd-group/12>;gp="lights1";d="domain1"

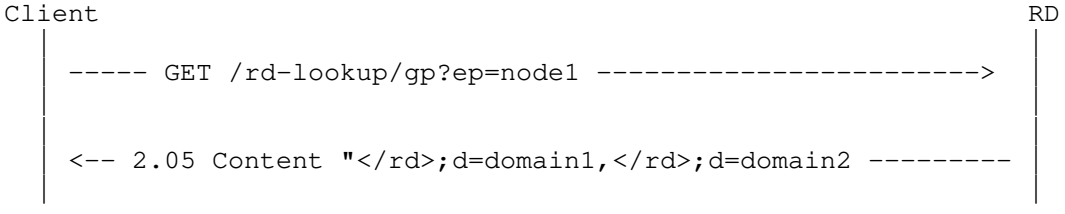
The following example shows a client performing a lookup for all endpoints in a particular group:



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:



```
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 collagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

12. Changelog

Changes from -04 to -05:

- Restricted Update to parameter updates.
- Added pagination support for the Lookup interface.
- Minor editing, bug fixes and reference updates.
- 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 inclusing 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: May 2, 2012

P. van der Stok
Philips Research
K. Lynn
Consultant

October 30, 2011

CoAP Utilization for Building Control
draft-vanderstok-core-bc-05

Abstract

This draft describes an example use of the RESTful CoAP protocol for building automation and control (BAC) applications such as HVAC and lighting. A few basic design assumptions are stated first, then URI structure is utilized to define group as well as unicast scope for RESTful operations.

This proposal supports the view that 1) service discovery is complementary to resource discovery and facilitates control network scaling, and 2) building control is likely to move in steps toward all-IP control networks based on the legacy efforts provided by DALI, LON, BACnet, ZigBee, and other standards.

The authority portion of the URI is used to identify a device (group) and the resulting DNS name is bound to a unicast (multicast) address. Group addressing has consequence for the naming convention of the resources of a device. Naming of URI is building or organization dependent, must be flexible, and SHOULD conform to some local convention. Naming of resources MUST be standardised preferrable by a building control related organisation.

It is shown that DNS-based service discovery can be used to locate URIs on the scale necessary in large commercial BAC deployments. The relation of DNS-SD and a Resource Directory is discussed. Finally, a method is proposed for mapping URIs onto legacy BAC resources, e.g., to discover application-layer gateways, proxies, and their dependent services.

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 2, 2012.

Copyright Notice

Table of Contents

1. Introduction

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 "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

In addition, the following conventions are used in this document:

A CoAP end-point, or server, is identified by a unique {IP address, port} tuple and characterised by a protocol. A server is completely specified by the authority part of a URI.

A device is the physical object that is connected to the network. A device may host one or more CoAP servers.

A service (in the service discovery sense) is a related set of resources on a CoAP server. A URI completely specifies the syntax of a service interface. Metadata describe the semantics of the service interface. The semantics may include the relation between service and the hardware connected to the device. A CoAP server may expose one or more services.

In examples below involving URIs, the authority is preceded by double slashes "//" and path is preceded by a single slash "/". The examples may make use of full or partial host names and the difference should be clear from the context.

1.2. Motivation

The CoAP protocol [I-D.ietf-core-coap] aims at providing a user application protocol architecture for a network of devices with a low resource provision such as memory, CPU capacity, and energy. In general, IT application manufacturers strive to provide the highest possible functionality and quality for a given price. In contrast, the building automation controls market is highly price sensitive and manufacturers tend to compete by delivering a given functionality and quality for the lowest price. In the first market a decreasing memory price leads to more software functionality, while in the second market it leads to a lower Bill of Material (BOM).

The vast majority of devices in a typical building control application is resource constrained, making the standardization of a lightweight application protocol like CoAP a necessary requirement for IP to penetrate the device market. The low energy consumption requirement of battery-less devices reinforces this approach. Low

resource budget implies low throughput and small packet size as for [IEEE.802.15.4]. Reduction of the packet size is obtained by using the header reduction of 6LoWPAN [RFC4944] and encouraging small payloads.

Several legacy building control standards (e.g. [BACnet], [DALI], [KNX], [LON], [ZigBee], etc.) have been developed based on years of accumulated knowledge and industry cooperation. These standards generally specify a data model, functional interfaces, packet formats, and sometimes a physical medium for data objects and function invocation. Many of these industry standards also specify proprietary transport protocols, necessitating expensive stateful gateways for these standards to interoperate. Many more recent building control network include IP-based standards for transport (at least to interconnect islands of functionality) and other functions such as naming and discovery. CoAP will be successful in the building control market to the extent that it can represent a given standard's data objects and provide functions, e.g. resource discovery, that these standards depend on.

From the above the wanted basic syntax properties can be summarized as:

- Generate small payloads.
- Compatible with legacy standards (e.g. LON, BACnet, DALI, ZigBee Device Objects).
- Service/resource discovery in agreement with legacy standards and naming conventions.

This submission defines an approach in which the payload contains messages with a syntax defined by legacy control standards. Accordingly, the syntax of the service/resource discovery messages encapsulates legacy control standard. The intention is a progressive approach to all-IP in building control. In a first stage standard IETF based protocols (e.g. CoAP, DNS-SD) are used for transport of control messages and discovery messages expressed in a legacy syntax. This approach enables the reuse of controllers based on the semantics of the chosen control standard. In a later stage a complete redesign of the controllers can be envisaged guided by the accumulated experience with all-IP control.

Two concepts, hierarchy and group, are of prime importance in building control, particularly in lighting and HVAC. Many control messages or events are multicast from one device to a group of devices (e.g. from a light switch to all lights in an area). The scope of a multicast command or discovery message determines the group of devices that is targeted. A group scope may be defined as link-local, as a tree maintained by an IP-multicast protocol, or an

overlay that corresponds to the logical structure of a building or campus and is independent of the underlying network structure. Techniques for group communication are discussed in [I-D.rahman-core-groupcomm].

As described in "Commercial Building Applications Requirements" [I-D.martocci-6lowapp-building-applications] it is typical practice to aggregate building control at the room, area, and supervisory levels. Furthermore, networks for different subsystems (lights, HVAC, etc.) or based on different legacy standards have historically been isolated from each other in so-called "silos". RESTful web services [Fielding] represent one possible way to expose functionality and normalize data representations between silos in order to facilitate higher order applications such as campus-wide energy management.

Consequently, additional group properties are:

- Devices may be part of one or more groups.
- Resources addressed by a group must be uniformly and consistently named across all targeted devices.

For clarity, this I-D limits itself to two types of applications: (1) M2M control applications running within a building area without any human intervention after commissioning of a given network segment and (2) maintenance oriented applications where data are collected from devices in several building areas by devices inside or outside the building, and humans may intervene to change control settings. This I-D compares commercial building solutions with solutions for the home.

2. URI structure

This I-D considers three elements of the URI: scheme, authority, and path, as defined in "Uniform Resource Identifier (URI): Generic Syntax" [RFC3986]. The authority is defined within the context of standard DNS host naming, while the path is valid in relation to a fully qualified domain name (FQDN) plus optional port (and protocol is implicit, based on scheme). An example based on [RFC3986] is: foo://host.example.com:8042/over/there?name=ferret#nose, where "foo" is the scheme, "host.example.com:8042" is the authority, "/over/there" is the path, "name=ferret" is the query, and "nose" is the fragment. Fragments are not supported in CoAP.

2.1. Scheme part

The CoAP URI scheme syntax is specified in section 6 of [I-D.ietf-core-coap] and is compatible with the "http" scheme specification [RFC2616]. The scheme is implicit from the perspective of the service, but it indicates the protocol used to access the service to potential clients.

2.2. Authority part

The authority part is either a literal IP address or a DNS name comprised of a local part, specifying an individual device or group of devices, and a global part specifying a (sub)domain that may reflect the logical hierarchical structure of the building control network. The result is said to be a fully qualified domain name (FQDN) which is globally unique down to the group or device level. An optional port number may be included in the authority following a single colon ":" if the service port is other than the default CoAP value. The authority resolves to a {IP-address, port} tuple. The IP-address may be either unicast or multicast. The authority therefore identifies an individual server or a named group of servers.

The CoAP spec [I-D.ietf-core-coap] states "When a CoAP server is hosted by a 6LoWPAN device, it SHOULD also support a port in the 61616-61631 compressed UDP port space defined in [RFC4944]." As shown below, DNS-SD [I-D.cheshire-dnsext-dns-sd] is a viable technique for discovering dynamic host and port assignments for a given service. However, the use of dynamic ports in URIs is likely to lead to brittle (non-durable) identifiers as there is no assurance that a CoAP server will consistently acquire the same dynamic port and different {IP-address, port} tuples conventionally represent different servers.

A building can be unambiguously addressed by it GPS coordinates or more functionally by its zip or postal code. For example the Dutch Internet provider, KPN, assigns to each subscriber a host name based on its postcode. Analogously, an example authority for a building may be given by: //bldg.zipcode-localnr.Country/ or more concretely an imaginary address in the Netherlands as: //bldg.5533BA-125a.nl/. The "bldg" prefix can specify the target device within the building. Arriving at the device identified by //bldg.5533BA-125a.nl, the receiving service can parse the path portion of the URI and perform the requested actions on the specified resource.

Buildings have a logical internal structure dependent on their size and function. This ranges from a single hall without any structure to a complex building with wings, floors, offices and possibly a

structure within individual rooms. The naming of the building control equipment and the actual control strategy are intimately linked to the building structure. It is therefore natural to name the equipment based on their location within the building. Consequently, the local part of the URI identifying a piece of equipment is expressed in the building structure. An example is: //light-27.floor-1.west-wing...

This proposal assumes a minimal level of cooperation between the IT and building management infrastructure, namely the ability of the former to delegate DNS subdomains to the latter. This allows the building controls installer to implement an appropriate naming scheme with the required granularity. For institutional real estate such as a college or corporate campus, the authority might be based on the organization's domain, e.g. //device-or-group.floor.wing.bldg.campus.example.com/. In cases where subdomain delegation is not an option, structure can still be represented in a "flat" namespace, subject to the 63 octet limit for a DNS label: //group1-floor2-west-bldg3-campus.example.com.

Most communication is device to device (M2M) within the building. Often a device needs to communicate to all devices of a given type within a given area of the building. For example a thermostat may access all radiator actuators in a zone. A light switch located at room 25b006 of floor one, expressed as: //switch0.25b006.floor1.5533BA-125a.nl/, might specify a command to light1 within the same room with //light1.25b006.floor1.5533BA-125a.nl/. This approach can lead to rather verbose URI strings in the packet, contrary to the small packet assumption. The question arises as to whether the syntax of the authority part needs to be standardized for building control. Given the naming flexibility provided by DNS, authority names for building control are more the concern of the building owner or the installer than a standardization concern.

2.3. Path part

The path identifies the addressable attributes of the service at the highest possible granularity. A set of paths defines the syntax of the service invocation and constitutes the interface description of the service. Every network service attribute is completely identified by a URI scheme://authority/path. In analogy, the path part of the URI specifies the resource of a given server. The naming of the services and their associated attributes are typically subjects for standardization. There is no widely accepted standard for uniformly naming building control services in a URI. A vigorous effort is undertaken by the oBIX working group of OASIS [oBIX], but its current impact is limited. There is also an open source point

naming effort underway called Project Haystack [HAYSTACK].

The path is constructed like a file system path name. It consists of a sequence of one or more name fields, with each field preceded with a slash, like /func1/subf2/final. The set of paths is structured as a tree. The last name in a name field sequence is called a leave of the tree, and the authority is the root of the path tree of a given host. The semantics of a given sub-tree in the path tree is specified by the Interface Description (if=) attribute described in [I-D.ietf-core-link-format]. As for file systems some tree naming with associated semantics can be standardized such as the de facto PC standard directory "documents and settings" with the sub-directories "My documents", "usradmin", etc. When a given body, e.g. XXX, has defined a name structure and semantics for the path tree, we say that "if = XXX" when the path tree conforms to the name structure defined by XXX.

When a GET method with an URI like "//t-sensor1.25b006.floor1.example.com/temperature" is sent, it represents an a priori understanding that the server with name t-sensor1 exists, provides a service of a given standard type (with associated semantics) (e.g. ZigBee temperature sensor), and that this standard type has the readable attribute: temperature. When commands are sent to a group of servers it MUST be the case that the targeted resource has the same path on all targeted servers. Therefore, it is necessary to establish at least a local uniform path naming convention to achieve this. One approach is to include the name of the standard, e.g. BACnet, as the first element in the path and then employ the standard's chosen data scheme (in the case of BACnet, /bacnet/device/object/property).

The organization responsible for defining a given industry standard XXX (e.g. BACnet, ZigBee, etc.) can register the /.well-known/XXX prefix and specify the allowable path-names for a server of a given type. The same body also defines the "if=XXX" attribute. This allows the standards development organization responsibile for XXX to define the name space and resources associated with the prefix together with the associated semantics. The registered /.well-known/ XXX URI effectively defines a standard object model, or schema, for services of the XXX application protocol. Manufacturers may optionally define proprietary resources that can be discovered dynamically using methods described below.

Although the authority part names need not always be transported, the path names MUST be transported in the CoAP packets. Therefore, path names SHOULD be as short as possible, even at the detriment of the clarity of the meaning of the path name.

3. Group Naming and Addressing

Within building control it is necessary to send the same command to a set of servers. Grouping allows to invoke the set of services with one application command to be executable within a specified time interval. Given a network configuration, the network operator needs to define an appropriate set of groups which can be mapped to the building areas. Knowledge about the hierarchical structure of the building areas may assist in defining a network architecture which encourages an efficient group communication implementation. IP-multicasting over the group is a possible approach for building control, although proxy-based methods may prove to be more appropriate in some deployments [I-D.rahman-core-groupcomm].

Example device groups become:

URI authority	Targeted group
//all.bldg6...	"all devices in building 6"
//all.west.bldg6...	"all devices in west wing, building 6"
//all.floor1.west.bldg6...	"all devices on floor 1, west wing, ..."
//all.bu036.floor1.west.bldg6...	"all devices in office bu036, ..."

The granularity of this example is for illustration rather than a recommendation. Experience will dictate the appropriate hierarchy for a given structure as well as the appropriate number of groups per subdomain. Note that in this example, the group name "all" is used to identify the group of all devices in each subdomain. In practice, "all" could name an address record in each of the DNS zones shown above and would bind to a different multicast address [RFC3596] in each zone. Highly granular multicast scopes are only practical using IPv6. The multicast address allocation strategy is beyond the scope of this I-D, but various alternatives have been proposed [RFC3306][RFC3307][RFC3956].

To illustrate the concept of multiple group names within a building, consider the definition, as done with [DALI], of scenes within the context of a floor or a single office. For example, the setting of all blue lights in office bu036 of floor 1 can be realized by multicasting a message to the group "//blue-lights.bu036.floor1". Each group is associated with a multicast IP address. Consequently, when the application specifies the sending of an "on" message to all blue lights in the office, the message is multicast to the associated IP address.

The binding of a group FQDN to a multicast address (i.e., creation of the AAAA record in the DNS zone server) happens during the

commissioning process. Resolution of the group name to a multicast address happens at restart of a device. A multicast address and associated group name in this context are assumed to be long-lived. It can happen that during operation the membership of the group changes (less or more lights) but its address is not altered and neither is its name. Group membership may be managed by a protocol such as Multicast Listener Discovery [RFC5790].

Similarly, a group can identify a set of resources of one server. For examples a device contains four I/O channels. The device hosts one server with four resources to access each of the four individual channels separately. Commonly, it is also required to access all four channels as one group. An additional path identifies the group of services. An example set of services and service-group is:

URI path	Targeted group
/IOchannel/1...	"channel 1 of the IO channel device "
/IOchannel/2...	"channel 2 of the IO channel device "
/IOchannel/3...	"channel 3 of the IO channel device "
/IOchannel/4...	"channel 4 of the IO channel device "
/IOchannel/...	"channel 1 to 4 of the IO channel device "

A group defines a set of servers possibly containing a set of resources. Grouping of the resources is provided by the device manufacturer. Grouping of the servers is supported by DNS and multicast protocols. The multicast address(es) identify the servers belonging to the group. A given server might belong to a number of groups. For example the server belonging to the "blue-lights" group in a given corridor might also belong to the groups: "whole building", "given wing", "given floor", "given corridor", and "lights in given corridor". From the perspective of a server, the main consequence of joining a group is it should accept packets for an additional IP address. The granularity of the domain names may have an impact on the complexity of the DNS infrastructure but not necessarily on the low-resource destinations or sources. Assuming that resolution of addresses only happens at device start-up, the complexity of the DNS server need not affect the responsiveness of the devices.

In summary, the authority portion of the URI resolves to an IP-address and port number, and identifies a server or group of servers. Authority naming is building or organization dependent, must be flexible, and does not require standardization efforts but SHOULD conform to some uniform convention. Path naming SHOULD conform to the naming convention of a standardization body.

4. Discovery

4.1. Service discovery goals

Service discovery in building control should rely on a minimal need for intervention by humans (or complete absence of humans) during system setup, bootstrapping, restart, configuration and daily operation. The goals for service discovery area:

Goal	Goal description
Return_instance	Return all instances of a given service type within a given domain
Group_instance	Group a set of instances within a group associated with a domain
Instance_resolution	Resolve the instance name to usable invocation information (e.g. IP address and port)
Group_resolution	resolve the group name to usable invocation information (e.g. IP address and port)

These goals are necessary to support the operation of commercial building control. Returning the instances results in a list of names. For building control these names can be any sequence of characters as long as for each service instance these names are unique within the domain. In [I-D.cheshire-dnsext-dns-sd] the office equipment in the IT domain is recommended to use understandable and human-readable names. The Home domain may have a need for human understandable names. This is not the case for the commercial building automation domain. However, uniqueness of the name is necessary for the application that needs to address the service in a consistent manner. Given the large number of devices in a building (several hundreds to thousands) scaling is an important aspect of the service discovery. A set of central DNS servers will provide the scalability. The expectation is that names need to be managed consistently by a central authority which can be supported by the DNS server. Tools will assist the installer and operator of the network to do the installation, configuration and maintenance of the network structure. Small devices will use the DNS server to learn the communication partners providing a given service within their domain and to resolve the IP addresses of the communication partners.

Within the home it is more important that the names convey the purpose of the service to the user reading the names and selecting his favored service instance. Non-unique names, although confusing, can probably be handled by the user of these names. Scalability is less of an issue because a smaller number of devices is implicated. The network in the home is probably more dynamic than its commercial counter-part, with many movements of devices and arrival or removal of devices.

Section 5 presents some examples of DNS structures to show how the choice of names influences the granularity of the discovery. In sections 5.1 and 5.3 a grouping example and a commissioning example, filling the DNS, are presented.

4.2. DNS-Based Service Discovery

DNS-Based Service Discovery (DNS-SD) defines a conventional way to configure DNS PTR, SRV, and TXT records to facilitate discovery of services within a subdomain, re-using the existing DNS infrastructure. This section gives a cursory overview of DNS-SD; see [I-D.cheshire-dnsext-dns-sd] for a complete description.

A DNS-SD service instance name is of the form <Instance>.<ServiceType>.<Location>.

The Location part of the service name is identical to the DNS subdomain part of the authority in URIs that identify the resources of this server or group and may identify a building zone as in the examples above.

The ServiceType SHOULD have the form [_subtype._sub.]_type._proto (e.g. _temp._sub._bc._udp). The _proto identifier provides a transport protocol hint as required by the SRV record definition [RFC2782] and, in the case of CoAP, it is always "_udp". The _type identifier is determined by standards development organization (SDO) and MUST be registered with dns-sd.org [dns-sd] (e.g. _bc for building control). The SDO is then free to specifiy one or more _subtype identifiers, which must be unique for a given _type (e.g. _temp). The _subtype and _type labels are separated by the literal "._sub" label.The maximum length of the type and subtype fields is 14 octets, but shorter names are encouraged to reduce packet sizes.

A PTR record with the label "_type._proto" is defined for each server in a selected domain, and this record's value is set to the service instance name (which in turn identifies the SRV and TXT records for the CoAP server).

The Instance part of the service name may be changed during the commissioning process. It must be unique for a given ServiceType within the subdomain. The complete service name uniquely identifies an SRV and a TXT record in the DNS zone. The granularity of a service name MAY be at the group or server level, or it could represent a particular resource within a CoAP server. The SRV record contains the host (AAAA record) name and port of the service. The path part of the URI MUST be placed in the TXT record (path=) when multiple resources belong to the same service.

4.3. Browsing for Services

Devices in a given Location with given ServiceType, _type._proto, may be enumerated by sending a DNS query for PTR records named _type._proto to the authoritative server for that zone associated with the Location. A list of instance names for SRV records matching that <ServiceType>.<Location> is returned. Each SRV record contains the host name and port of a CoAP server. The IP address of the device 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. Apart from defining standardized resources identified by if=XXX, the XXX organization may also define the standard "key=value" pairs present in the TXT record, e.g. type=switch. By convention, the first pair is txtver=<number> so that different versions of the XXX schema may interoperate. For example: A query is sent to DNS-SD to return all DALI lamps within the domain office5/mybuilding and with ServiceType: _lamp._sub._dali._udp. DNS-SD returns the list of all SRV records and AAAA records of the devices within the domain providing the wanted service.

4.4. Resource vs Service Discovery

Service discovery is concerned with finding the IP address, port, protocol, and possibly path of a named service. Resource discovery is a fine-grained enumeration of resources (path-names) of a server. [I-D.ietf-core-link-format] specifies a resource discovery pattern, such that sending a confirmable GET message for the /.well-known/core resource returns a set of links available from the server. These links describe resources hosted on that server.

CoAP link format can be used to enumerate attributes and populate the DNS-SD database in a semi-automated fashion. CoAP resource descriptions can be imported into DNS-SD for exposure to service discovery as described in [I-D.lynn-core-discovery-mapping]. The values stored in the DNS-SD directory are extracted from the information stored in the resource directory associated with a set of CoAP hosts [I-D.shelby-core-resource-directory]. The resources describe how the services can be manipulated in detail and in concreto.

It is assumed that a resource directory exists per 6LoWPAN [RFC4944], possibly running on the edge router. The DNS-SD provides a larger scope by storing the info of all services over a set of interconnected 6LoWPANs. Where the resource directory is possibly completely adequate for home networks, handling of multiple resource directories can be quite cumbersome for the many 6LoWPANs envisaged for offices. However, during network configuration, the resource

directory can be used as long as the DNS is not yet accessible.

The DNS-SD approach is complementary to the more fine-grained resource discovery, fits better the concept of service by discovering servers with given properties. DNS-SD supports a hierarchical approach to the naming of the services as discussed in section 3. DNS-SD provides a directory structure that scales well with the network size as shown by its present-day operation.

5. DNS record structure

An example is presented which explains the Resource Record (RR) structure on the DNS server. This section follows the mapping specified in [I-D.lynn-core-discovery-mapping], which defines how to fill the DNS-SD records from the link extension values. Suppose the services are delivered by XXX building control devices. The example subtype- and context- names are assumed to be standardized by the XXX alliance. All devices are situated in one office with location office4.bldg8.example.com. The names in the examples are more verbose than recommended to make the examples more readable. The table presents the services provided in the office control network:

service	ServiceType	Number
illumination	_OnOff_light._sub._bc._udp	4
presence	_occup_sensor._sub._bc._udp	1
temperature	_temp_sensor._sub._bc._udp	1
shading	_shade_control._sub._bc._udp	1

In DNS PTR records with as label the ServiceType have as value service instance names. The unique Instance names identify the service instances. In the example, the names contain id-x, with x in natural numbers. The names are usually created at the factory floor and somehow attached to the product. The ServiceTypes have been suffixed with .04.b8 to represent office4 in building8. The same suffix is used as PTR label to enemerate all instance of a given service, or within a given domain.

```
_OnOff_light._sub._bc._udp.04.b8   PTR id-1._OnOff_light
bc._udp.04.b8                      PTR id-1._OnOff_light
04.b8                              PTR id-1._OnOff_light
_OnOff_light._sub._bc._udp.04.b8   PTR id-2._OnOff_light
bc._udp.04.b8                      PTR id-2._OnOff_light
04.b8                              PTR id-2._OnOff_light
_OnOff_light._sub._bc._udp.04.b8   PTR id-3._OnOff_light
bc._udp.04.b8                      PTR id-3._OnOff_light
04.b8                              PTR id-3._OnOff_light
_OnOff_light._sub._bc._udp.04.b8   PTR id-4._OnOff_light
bc._udp.04.b8                      PTR id-4._OnOff_light
04.b8                              PTR id-4._OnOff_light
_occup_sensor._sub._bc._udp.04.b8  PTR id-5._occup_sensor
bc._udp.04.b8                      PTR id-5._occup_sensor
04.b8                              PTR id-5._occup_sensor
_temp_sensor._sub._bc._udp.04.b8   PTR id-6._temp_sensor
bc._udp.04.b8                      PTR id-6._temp_sensor
04.b8                              PTR id-6._temp_sensor
_shade_control._sub._bc._udp.04.b8 PTR id-7._temp_sensor
bc._udp.04.b8                      PTR id-7._temp_sensor
04.b8                              PTR id-7._temp_sensor
```

In the above example the id-x identifiers without the subtype suffix would be discriminating enough.

Discovery can be done with the following results. A query with the following argument returns

```
query argument                     result list
.04.8                              id-1._OnOff_light
                                   .......
                                   id-7._temp_sensor
_bc._udp.04.b8                     id-1._OnOff_light
                                   .......
                                   id-7._temp_sensor
_OnOff_light._sub._bc._udp.04.b8   id-1._OnOff_light
                                   .......
                                   id-4._OnOff_light
_occup_sensor._sub._bc._udp.04.b8  id-5._occup_sensor
```

When other offices are included in the database, the query argument 04.b8 selects those entries which are associated with office4 in building8 and rejects any others. The example shows clearly the query granularity that can be obtained and the care that must be exercised when defining the names of the ServiceTypes.

The service instances (value of PTR records) are the labels of the SRV, AAAA and TXT records describing the service instance. The SRV

record specifies the location (authority) and the port number. In the authority o4.b8 refers to office4 in building8. The AAAA record specifies the IP-address, while the TXT record specifies the subtype and the data representation of the legacy parser (if = ZigBee).

```
id-1._OnOff_light    SRV  light1.o4.b8.example.com Port-x
                     AAAA fdfd::1234
                     TXT  if=ZigBee
id-2._OnOff_light    SRV  light2.o4.b8.example.com Port-x
                     AAAA fdfd::1235
                     TXT  if=ZigBee
id-3._OnOff_light    SRV  light3.o4.b8.example.com Port-x
                     AAAA fdfd::1236
                     TXT  if=ZigBee
id-4._OnOff_light    SRV  light4.o4.b8.example.com Port-x
                     AAAA fdfd::1237
                     TXT  if=ZigBee
id-5._occup_sensor   SRV  occup.o4.b8.example.com  Port-x
                     AAAA fdfd::1238
                     TXT  if=ZigBee
id-6._temp_sensor    SRV  temp.o4.b8.example.com   Port-x
                     AAAA fdfd::1239
                     TXT  if=ZigBee
id-7._shade_control  SRV  shade.o4.b8.example.com  Port-x
                     AAAA fdfd::1240
                     TXT  if=ZigBee
```

It is possible that the temperature sensor and occupancy sensor are delivered on one device. The consequence is that one device hosts two services. In the DNS table the four lights and the shade controller are unaffected. However, the PTR records with the occupancy and temperature sensor point to the same unique identifier id-8 that is suffixed with the name of the subtype. This example shows that the subtype suffix is needed to discriminate between the two service instances.

```
_occup_sensor._sub._bc._udp PTR id-8._occup_sensor
_temp_sensor._sub._bc._udp  PTR id-8._temp_sensor
```

Two SRV records with accompanying AAAA and TXT records describe the two servers, each providing one service, in more detail. The servers share the same IP address but are connected to different ports, and do have a different paths names. The TXT record is used to specify the path part with "path=".

```
id-8._occup_sensor SRV  occup.o4.b8.example.com Port-x
                   AAAA fdfd::1241
                   TXT  path=/os if=ZigBee
id-8._temp_sensor  SRV  temp.o4.b8.example.com  Port-y
                   AAAA fdfd::1241
                   TXT  path=/ts if=ZigBee
```

The path names /ts and /os are short names for temperature_sensor and occupancy_sensor respectively. Not all multi-function devices will use different ports for the individual functions. It is also quite common to use different IP interfaces with different IP addresses, reflected by the value of the AAAA records.

5.1. DNS group example

Another aspect is the grouping of servers. Where in the former section the names of the services are standardized names, this is less probable for the group names. Usually the group names are application specific or are standardized at the manufacturer. For example, assume that a group all-light.o4.b8.example.com is created which contains all four lights inside office4. The accompanying ServiceType can be defined as _all_light._sub._bc._udp. The ServiceType suffixed with 04.b8 points to a unique identifier defined as _all_light.04.b8, assuming that this is the only _all_light group within office 4 of building 8. The PTR record looks like:

```
_all_light._sub._bc._udp.04.b8 PTR _all_light.04.b8
```

It is assumed that the group all_light.o4.b8.example.com has received a multicast address: ff1e::148. The accompanying SRV, AAAA, and TXT RR become:

```
_all_light.04.b8 SRV  all_light.o4.b8.example.com Port-z
                 AAAA ff1e::148
                 TXT  if=ZigBee
```

When a multicast message is sent to a group, the path of the accessed resource must be strictly the same for all servers. The naming of the path is typically a responsibility for the standardisation organisations describing the command set for a given application area. However a constraint exits in the case of mult-function devices which host multiple resource of the same type. For example a device with three lamps with corresponding onoff attributes can be accessed via the three different paths:

```
                              /light/1/onoff
                              /light/2/onoff
                              /light/3/onoff
```

A unique path to the onoff resource of all instances of light on this device can be provided by /light/onoff. As this is logically the path to a single instance on a mono-function device. The corresponding unique paths for onoff to be used in the multicast message becomes /light/onoff. The corresponding resource records for a luminaire, named lm1, in DNS become:

```
 _light._sub._bc._udp.04.b8 PTR  _all_light.04.b8
 _light._sub._bc._udp.04.b8 PTR  _light_1.04.b8
 _light._sub._bc._udp.04.b8 PTR  _light_2.04.b8
 _light._sub._bc._udp.04.b8 PTR  _light_3.04.b8
 _all_light.04.b8           SRV  all_light.o4.b8.example.com Port-x
                            AAAA ff1e::148
                            TXT  if=ZigBee path=/light
 _light_1.04.b8             SRV  lm1.o4.b8.example.com       Port-z
                            AAAA fdfd::1234
                            TXT  if=ZigBee path=/light/1
 _light_2.04.b8             SRV  lm1.o4.b8.example.com       Port-z
                            AAAA fdfd::1234
                            TXT  if=ZigBee path=/light/2
 _light_3.04.b8             SRV  lm1.o4.b8.example.com       Port-z
                            AAAA fdfd::1234
                            TXT  if=ZigBee path=/light/3
```

The entries in DNS can be used to form groups with the light weight group management protocol and multicast listener discovery [RFC5790].

5.2. Operational use of DNS-SD

The populated DNS-SD server provides the necessary support for the applications to execute their control loops with minimum operator support. The operation of the office network can be split up in phases. In a first phase the network is commissioned, such that a relation is established between the IP address, the servicetype and the domain. The servicetype can be extracted from the link-format as described in [I-D.shelby-core-resource-directory]. After commissioning this information is stored in the DNS-SD files. In a second phase groups are formed and group names with their IP address are stored in the DNS-SD files. The IP multicast addresses are communicated to the members of the groups. In the third and final phase, applications query DNS-SD to find the IP addresses of the services within a given domain, and of the groups within a given domain.

In the home, a commissioning phase requiring the intervention of an installer (a "truck roll") is to be avoided if possible. Here the first phase consists of the booting up devices which insert their services resources to a link-format directory. The information from the resource directory can be inserted into DNS-SD or into xmDNS [I-D.lynn-dnsext-site-mdns] when appropriate. In the second phase remote controllers or other hand-held devices can be used to discover the services of a given type, to group the services, and to store the group names into DNS-SD or xmDNS as appropriate. Pointing out the members of a group can be in any kind of manner from typing members in, selecting them from a browser list, etc.

5.3. Commissioning CoAP devices

For clarity it is assumed in this section that a device hosts one server. A device has received a unique device identifier at the production plant. Given the authority naming presented in section 2.2 the authority name represents the location of the host within the building.

Commissioning means the following three actions:

- Defining the URI (location)
- Assigning an IP address to the URI
- mapping the unique device identifier to the URI

Two cases of the office network are considered for commissioning: (1) no 6LBR and no DNS server connected, and (2) a 6LBR connects the office network to a DNS server.

When an architect has designed the building and described all light points, ventilators, heating- and cooling units, and sensors, it is necessary to identify all these devices spatially and functionally. Storing the triple <Instance>.<ServiceType>.<Location> into DNS-SD represents the commissioning process. The Instance is the unique identifier given to the device in the factory but which has no relation to its later location. The ServiceType together with the Location represent the spatial and functional aspects of the device as specified by the architect.

Design decision: A commissioning tool with access to the network is used for the commissioning phase.

For example, dependent on used technology and production process, the following situation (state) may exist in a host after physical installation of the devices and before commissioning:

- A given host is unaware of its Location.

- A given host knows its ServiceType and Instance. The Instance is also readable by bar code reader.

- The commissioning tool knows all Locations to which hosts need to be assigned.

- Each host has a (site-local) IP address.

Consider the commissioning process (1) with a central DNS-SD server and (2) without a central server using xmDNS. The commissioning processes described below are just examples and should not be taken as working procedures for commissioning devices in a building.

5.3.1. DNS-SD server present

The installer reads with a bar code reader, attached to the commissioning tool, the identifier of the device to commission. It is assumed that the tool can learn the IP address of the device with the given identifier. The tool displays on a screen the physical lay-out of the devices within the building. The installer selects, on the screen of the tool, the physical location of the chosen device. From the designated physical location the tool creates the URI of the selected device. The tool inserts the URI and the IP address into the DNS server. For example the light with URI light1.o4.b8.example.com is represented with an AAAA record:

light1.o4.b8.example.com AAAA fdfd::1234

The tool reads the service name and type from the device using resource information stored according to the link-format [I-D.ietf-core-link-format]. With this information the tool constructs the PTR, SRV and TXT records according to the example presented in section 5.

This is done for all devices within a given part of the building. After the commissioning process, all resources of each device have an URI and IP address which are stored in the central DNS-SD server. When devices are restarted, the DHCP server may allocate new IP addresses to the device and update the DNS server.

5.3.2. DNS-SD server not present

It is assumed that the building network is composed of independent network segments (possibly a single site) such that each device on a given segment can communicate directly with any other device on this segment. The segments are not connected to a 6LBR and have no access

to DNS or other servers. The installer knows these segments and has a list of devices for a given segment. In the tool the installer selects the names which belong to the given building segment. The selected names are converted to site-local authorities and stored in the tool. All devices are assumed to have selected a site-local IP address. Assume that every device has a unique barcode within the building and that the corresponding device knows the bar code number. The installer reads with a bar code reader, attached to the tool, the Instance name of the device to commission. The installer selects, on the screen of the tool, the physical location of the chosen device. The tool knows the authority of the selected device. The tool broadcasts the bar code number and authority to all connected devices. The device with the given barcode number, extends the authority with the path name of the resources. For each resource, the device multicasts the site-local IP-address and the site-local URI to the xmDNS servers in the connected devices. This concludes the commissioning of a network segment. All resources of each device have a site-local URI and a site-local IP address which are stored in the xmDNS servers.

5.4. Proxy discovery

Proxies will be used in CoAP networks for at least two major reasons: (1) http/coap proxy, and (2) proxy of service on battery-less device. The first proxy is probably implemented as forward proxy, while the latter is probably implemented as backward proxy. The battery-less device will at rare occasions (when it is not sleeping) and during installation answer the GET /.well-known/core request. The return data are used by the installation tool to make the proxy device return the same resource names on /.well-known/core as is returned by the sleeping device. An installation tool installs on the proxy all the resources of the sleeping device for which the proxy is assumed to answer. Consequently, the proxy is discovered as a multi-server host with as many path names as it proxies sleeping servers. The servers on sleeping devices should not be discoverable via DNS-SD. However, AAAA records are generated for the sleeping device host name. This host name is used by the proxy to subscribe to the "sporadic" services of the sleeping device. For example assume two sleeping devices, an occupancy sensor and a temperature sensor, and one proxy. Two service types are defined with PTR records in DNS-SD. The identifier id-1 of the proxy is used by the installation tool to define the Instances.

```
_occup_sensor._sub._bc._udp.04.b8 PTR id-1._occup_sensor
_temp_sensor._sub._bc._udp.04.b8  PTR id-1._temp_sensor
```

Two SRV records with accompanying AAAA and TXT records describe the two services in more detail. The services share the same IP address,

are connected to the same port, but do have different paths names. The TXT record is used to specify the path part with "path=".

```
id-1._occup_sensor      SRV  proxy.o4.b8.example.com Port-x
                        AAAA fdfd:: 1241
                        TXT  path=/os if=ZigBee
id-1._temp_sensor       SRV  proxy.o4.b8.example.com Port-x
                        AAAA fdfd:: 1241
                        TXT  path=/ts if=ZigBee
sl-ts.o4.b8.example.com AAAA fdfd::1242
sl-os.o4.b8.example.com AAAA fdfd::1243
```

The path names /ts and /os are short names for temperature_sensor and occupancy_sensor respectively and were taken over from link-format information contained in the sleeping devices. Two AAAA records are provided for the two sleeping devices. The proxy has used the domain names of the sleeping devices to subscribe to the publications of the two sleeping devices.

It is important to remark that there are now two services with the same resources present on two different devices: the sleeping device and its proxy. When a host invokes the /.well-known/core resource, it should be possible to distinguish between the proxy (to be invoked) and the sleeping device (not to be invoked). The distinction is necessary once the sleeping device is discoverable and the sleeping device is awake from time to time. It is suggested that the link-format syntax allows to make this distinction.

6. Legacy data Representations in CoAP

Before CoAP devices can come to market, manufacturers must agree that the type and resources of the device can be interpreted according to some generally recognized syntax. At this moment no such generally recognized syntax exists for CoAP devices. We do not expect an IETF working group to standardize such a syntax, and we are convinced that syntax standardization is the responsibility of industry standards organizations. Given the long history of building control, many groups have defined a data representation for building control devices for example BACnet, ZigBee, oBIX, LON, KNX, and many others. It is our belief that new representations will be defined and must coexist with the named legacy ones.

The CoAP protocol should transport any data representation, and certainly the legacy ones.It is expected that a CoAP client can handle one or more legacy representation. Given that a CoAP client can handle representation of standard XXX, this I-D proposes that such a CoAP device can communicate with legacy devices via a CoAP/

legacy gateway (router).

6.1. Network architectures

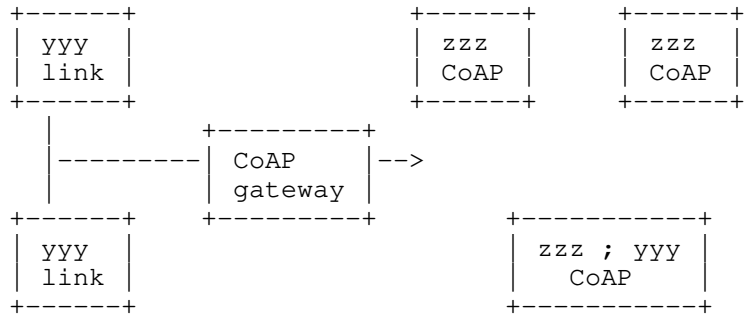


Figure 1: network with multiple representation standards

Figure 1 represents the network architecture which is expected for the purpose of this I-D. The CoAP gateway connects one link with two legacy devices -containing legacy data representation "yyy"- with the wireless CoAP network composed of three CoAP hosts. Two CoAP hosts contain the CoAP stack with a zzz representation and one host contains the CoAP stack with a zzz and an yyy representation. The yyy hosts can freely communicate according to the yyy link protocol over the yyy link. The zzz CoAP hosts, including the zzz;yyy host can freely exchange zzz data representations according to the CoAP protocol over the wireless 6LoWPAN network. The zzz;yyy host can send yyy data representations to the CoAP gateway which passes them on to the specified yyy legacy host. The yyy legacy device returns data to the requesting zzz;yyy CoAP host via the same gateway.

The CoAP hosts can address the legacy devices behind the gateway in at least 4 ways.

- All devices of legacy network YYY share the URI with the CoAP gateway. Every legacy device is a resource for the gateway as seen from the CoAP host. Consequently, the CoAP host sends the message to the IP address of the gateway and the gateway parses the URI-Path to determine the specified legacy device.

- All devices of legacy network YYY have IP addresses different from the IP address of the gateway. Consequently, a CoAP host sends the message to the IP address of the specified device. The routing protocol on the CoAP network makes the message arrive at the CoAP gateway. The gateway determines the specified legacy device from the destination IP address.

- All devices of legacy network YYY have different authorities. The authorities of the legacy device resolve to an IP address of the gateway. This means that the possibly lengthy authority names need to be transmitted. The gateway recognizes the authorities and maps authority to legacy device.

- All devices of legacy network YYY have different ports. This can be expressed in two ways (1) as :port in the URI, or (2) in the DNS-SD records. In the latter case the port is defined in the UDP header and is efficient in packet header size.

The major advantage of all four approaches is that the gateway only handles the URI or IP address and port number to select the destination legacy device independent of the type of legacy device and the contents of the legacy payload of the message. In Figure 1 the gateway connects to a single link. For example, this would be the case for DALI standard. Other legacy standards, like BACnet, LON, allow networks composed of multiple links.

An example of an invocation of a ZZZ service (See figure 2). The resource path /ZZZ identifies the parser of the ZZZ syntax. A 12 octet string completely describes the ZZZ command. The host is completely identified by the authority in the URI. The ZZZ parser on the host is identified by the port number in the UDP header (not shown).

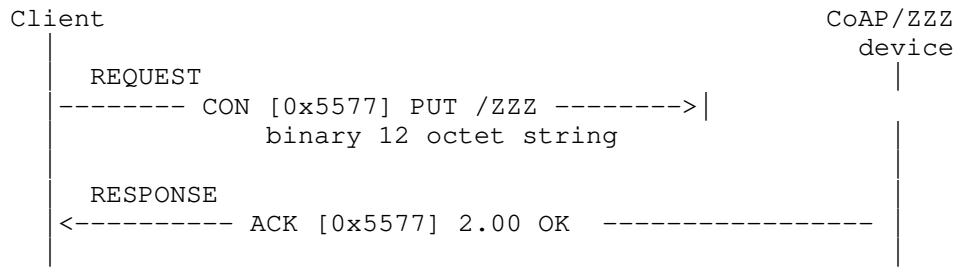


Figure 2: Sending a ZZZ command with CoAP to CoAP/ZZZ device

An example of an invocation of a DALI legacy device behind a gateway is given in figure 3. The resource path /DALI identifies the DALI parser. The application sets a value of 200 in the DALI device in the resource 256 defined by the DALI spec.

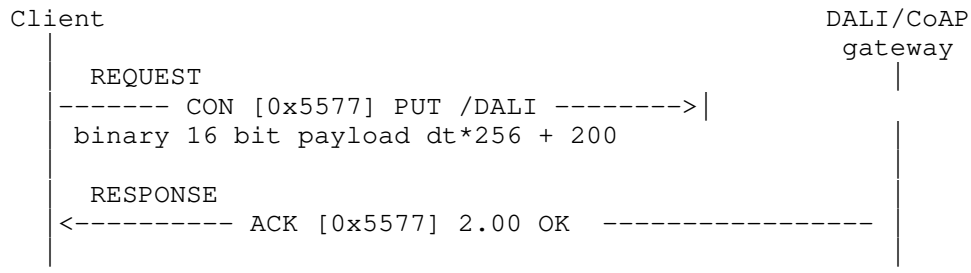


Figure 3: Sending a DALI setting with CoAP to CoAP/DALI gateway

6.2. Discovery of legacy gateways

Discovery of legacy gateways is not very different from discovery of proxies in section 5.4. the consequences for discovery are listed for the four modes of addressing legacy devices via a gateway of section 6.1.

- The gateway presents a list of resources representing the legacy devices. Discovery is done as for other CoAP devices.

- Each legacy device has a different IP address. The gateway must create entries in the DNS for as many legacy devices. The authority of the legacy device is the authority of the gateway with a ServiceType to be specified by the gateway.

- All devices of legacy network YYY have different authorities. In this case each legacy device has the same IP address as the gateway. The gateway must create entries in the DNS for as many legacy devices.

- All devices of legacy network YYY have different ports. The gateway must create entries in the DNS for as many legacy devices. Each entry has the authority of the gateway with a different ServiceType and a different port number.

7. Conclusions

This I-D explains how naming in building control is based on a hierarchical structure of the building areas. It is shown that DNS naming can be used to express this hierarchy in the authority portion of the URI, down to the group or device level. The hierarchical naming scheme need not be standardized, but rather can be designed to suit the application. However, it is recommended that the scheme be employed consistently throughout the delegated subdomain(s).

The authority portion of the URI is resolved by the client, using conventional DNS, into the unicast or multicast IP address of the targeted device(s). Taking advantage of the CoAP design [I-D.ietf-core-coap], the URI-Host option need not be transmitted in requests to origin servers and thus there is no performance penalty for using descriptive naming schemes. The CoAP design allows sending a short URI to distinguish between resources on a given device, resulting in very compact identifiers.

DNS-SD [I-D.cheshire-dnsext-dns-sd] can be used to scale up service discovery beyond the 6LoWPAN. DNS-SD can be used to enumerate instances of a given service type within a given sub-domain. This affords additional flexibility, such as the ability to discover dynamic port assignments for CoAP device, locate CoAP devices by subtype, or bind service names for particular CoAP URIs.

This I-D discusses the addressing, discovery and naming of legacy devices behind gateways. The discovery of backward proxies of sleeping devices is handled in a similar fashion.

A targeted resource is specified by the path portion of the URI. Again, this I-D does not mandate a universal naming standard for resources but uses examples to show how resources could be named for various legacy standards. An obvious requirement for resources that are accessed by multicast is that they MUST all share the same path. It is shown that it is possible to transport legacy commands (e.g. expressed in BACnet, LON, DALI, ZigBee, etc.) inside a CoAP message body. Entering ServiceTypes particular to a given standard necessitates that the standardization body declares the ServiceType to dns.org.

8. Security considerations

TBD: The detailed CoAP security analysis needs to encompass scenarios for building control applications.

Based on the programming model presented in this I-D, security scenarios for building control need to be stated. Appropriate methods to counteract the proposed threats may be based on the work done elsewhere, for example in the ZigBee over IP context.

Multicast messages are, by their nature, transmitted via UDP. Any privacy applied to such messages must be block oriented and based on group keys shared by all targeted devices. The CoRE security analysis must be broadened to include multicast scenarios.

9. IANA considerations

This I-D proposes that associations which standardize device representations (like BACnet, ZigBee, DALI,...) contact IANA to reserve the prefix /.well-known/XXX for the standard XXX.

10. Acknowledgements

This I-D has benefited from conversations with and comments from Andrew Tokmakoff, Emmanuel Frimout, Jamie Mc Cormack, Oscar Garcia, Dee Denteneer, Joop Talstra, Zach Shelby, Jerald Martocci, Anders Brandt, Matthieu Vial, Jerome Hamel, George Yianni, and Nicolas Riou.

11. Changelog

From bc-01 to bc-02

- Removed all references to multicast and multicast scope, given draft of rahman group communication.

- Adapted examples to CoAP-2 and core-link drafts.

- transport short URL for destination recognition.

- Elaborated legacy discovery under DNS-SD.

From bc-02 to bc-03

- Elaboration on gateways, commissioning and legacy networks.

- Recommendation to extend DNS-SD naming with sn, st, and ss attributes.

From bc-03 to bc-04

- moved core link extension sub-section to discovery mapping draft

- extended use of service type

- gave DNS record examples and worked out multifunction device

- added proxy discovery and legacy gateway discovery

- defined path tree and corresponding schema

- reviewed definition of group, device, server, service (interface),

resopurce, and attribute.

From bc-04 to bc-05

- extended and corrected examples for multi-function devicesw

- syntax more compatible with other resource discovery I-Ds

- abstract adapted

- more stringent use of the words server, end point, service and devices

12. References

12.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[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.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, August 2002.

[RFC3307] Haberman, B., "Allocation Guidelines for IPv6 Multicast Addresses", RFC 3307, August 2002.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", RFC 3596, October 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3956] Savola, P. and B. Haberman, "Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address", RFC 3956, November 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

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

[RFC5790] Liu, H., Cao, W., and H. Asaeda, "Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols", RFC 5790, February 2010.

12.2. Informative References

[I-D.cheshire-dnsext-dns-sd]
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-10 (work in progress), February 2011.

[I-D.cheshire-dnsext-multicastdns]
Cheshire, S. and M. Krochmal, "Multicast DNS", draft-cheshire-dnsext-multicastdns-14 (work in progress), February 2011.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-07 (work in progress), July 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-07 (work in progress), July 2011.

[I-D.martocci-6lowapp-building-applications]
Martocci, J., Schoofs, A., and P. Stok, "Commercial

Building Applications Requirements", draft-martocci-6lowapp-building-applications-01 (work in progress), July 2010.

[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-resource-directory]
Shelby, Z. and S. Krco, "CoRE Resource Directory", draft-shelby-core-resource-directory-01 (work in progress), September 2011.

[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-dnsext-site-mdns]
Lynn, K. and D. Sturek, "Extended Multicast DNS", draft-lynn-dnsext-site-mdns-01 (work in progress), March 2011.

[BACnet] Bender, J. and M. Newman, "BACnet/IP", Web http://www.bacnet.org/Tutorial/BACnetIP/index.html, 2000.

[ZigBee] Tolle, G., "A UDP/IP Adaptation of the ZigBee Application Protocol", draft-tolle-cap-00 (work in progress), October 2008.

[LON] "LONTalk protocol specification, version 3", 1994.

[DALI] "DALI Manual", Web http://www.dali-ag.org/c/manual_gb.pdf, 2001.

[KNX] Kastner, W., Neugschwandtner, G., and M. Koegler, "AN OPEN APPROACH TO EIB/KNX SOFTWARE DEVELOPMENT", Web http://www.auto.tuwien.ac.at/~gneugsch/fet05-openapproach-preprint.pdf, 2005.

[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 (LR-WPANs)", IEEE Std 802.15.4-2006, June 2006, <http://standards.ieee.org/getieee802/802.15.html>.

[HAYSTACK] "Project Haystack", Web http://project-haystack.org/, 2011.

[oBIX] Frank, B., Ed., "oBIX working group", Web http://www.obix.org, 2006.

[Fielding] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures, Second Edition", Doctoral dissertation , University of California, Irvine , Web http://www.ics.uci.edu/~fielding/pubs/dissertation/top.html, 2000.

[ZigBee-IP] "ZigBee Smart Energy Profile 2.0 Application Protocol Specification", Draft ZigBee-11167, March 2011.

[dns-sd] "dns-sd servicetype registration", Web http://www.dns-sd.org/ServiceTypes.html, 2011.

Authors' Addresses

Peter van der Stok
Philips Research
High Tech Campus 34-1
Eindhoven, 5656 AA
The Netherlands

Email: peter.van.der.stok@philips.com

Kerry Lynn
Consultant

Phone: +1 978 460 4253
Email: kerlyn@ieee.org