Abstract

This short I-D makes a number of partially interrelated proposals how to solve certain problems in the CoRE WG’s main protocol, CoAP.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 2, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

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

- 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.

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

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14 [RFC2119] and indicate requirement levels for compliant CoAP implementations.
2. A Compact Accept Option

A resource may be available in a number of representations. Without some information from the client, a server has no easy way to decide which of these would be best served. HTTP has an Accept: request header that a client can use to indicate the media types supported, allowing the server to decide. This header is somewhat unpopular as, for a web browser, there are too many media types to choose from, so even with wildcards -- there is no meaningful information to put there. (This has changed a bit for AJAX calls, which may indeed have a specific media type preference.) It is unlikely that machine-to-machine communication would have the same problem.

A similar function to the HTTP Accept: header could be added to CoAP as an option in a much simpler way. The CoAP Accept option would simply carry a value that is a sequence of octets, each of which is an acceptable media type for the client, in the order of preference (see Figure 1). If no Accept option is given, the client does not express a preference.

Accept also has to be given an option type code, e.g. 7, in Table 2 of [I-D.ietf-core-coap].

The other addition that would be required is an error code that mirrors HTTP's "415 Unsupported Media Type". This is indeed already listed as CoAP Code 35 in Table 3 of [I-D.ietf-core-coap].

Proposal: Add an Accept Option.

Benefits: A Server does not need to specify one URI each for every possible media type that it wants to serve a resource under.
Open Issues: For coap-00, this would have needed a way to handle two-byte media types (easiest if these can be made self-describing, at the cost of about 3 bits in the sub-type field; Figure 2).

An self-describing representation for long mediatypes could look like this:

```
0 0 1 2 3 4 5 6 7
+----------------
| top   | sub   | (1-byte: unchanged)
+----------------

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+--------------------------------
| 000  | top  | sub   | (2-byte)
+--------------------------------
```

Figure 2: A self-describing media type representation

Instead, we assume for now that CoAP-01 will switch to a single-byte media type encoding.
3. Representing Durations

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

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

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

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

3.1. Pseudo-Floating Point

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

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

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

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

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

For the purposes of this draft, we define an \((n,e)\)-pseudo-FP as a fixed-length value of \(n\) bits, \(e\) of which may be used for an exponent. Figure 3 illustrates an \((8,4)\)-pseudo-FP value.

```
+---+---+---+---+---+---+---+---+
| 0...          value           |
+---+---+---+---+---+---+---+---+
```

```
+---+---+---+---+---+---+---+---+
| 1... mantissa | exponent |
+---+---+---+---+---+---+---+---+
```

Figure 3: An \((8,4)\) pseudo-FP representation

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

The \((n,e)\)-pseudo-FP format can be decoded with a single line of code (plus a couple of constant definition), as demonstrated in Figure 4.

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

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

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

Only non-negative numbers can be represented by this format. It is designed to provide full integer resolution for values from 0 to \(2^{(n-1)}-1\), i.e., 0 to 127 in the \((8,4)\) case, and a mantissa of \(n-e\) bits from \(2^{(n-1)}\) to \((2^{n-2*e})*2^{(2^e-1)},\) i.e., 128 to 7864320 in the \((8,4)\) case. By choosing \(e\) carefully, resolution can be traded against range.
Note that a pseudo-FP encoder needs to consider rounding; different applications of durations may favor rounding up or rounding down the value encoded in the message. This requires a little more than a single line of code (which is left as an exercise to the reader, as the most efficient expression depends on hardware details).

3.2. A Duration Type for CoAP

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

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

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

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

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

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

Open Issues: It might be worthwhile to reserve one duration value, e.g. 0xFF, for an indefinite duration.
4. URI encoding

In HTTP-based systems, URIs can reach significant lengths. While CoAP-based systems may be able to sidestep the most egregious excesses (mostly by simply applying REST principles), a URI such as 

```
/.well-known/resources
```

can use up one third of the available payload in a CoAP message transported in a single 6LoWPAN packet. Is there a way to encode these URIs in a more efficient way?

Several proposals have been made on the CoRE mailing list, e.g. applying the principle of base64-encoding [RFC4648] in reverse and using only 6 bits per character. However, due to rounding errors and occasional characters that are not in the 64-character subset chosen to be efficiently encodable, the actual gains are limited. Similarly, using 7 bits per character (assuming URIs contain only ASCII characters) only gives a best-case gain of 12.5 %, and that only in the case the URI is a multiple of 8 characters long. On the other hand, the complexity (and danger of subtle interoperability problems) is not entirely trivial.

Appendix A.1 defines a potential URI encoding that is slightly more efficient than the abovementioned ones. However, even that was rejected by the WG for its unconvincing cost-benefit ratio, which then went on to discuss Henning Schulzrinne’s proposal to add state.

4.1. Stateful URI compression

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

Henning Schulzrinne has proposed for a server to be able to supply a shortened URI once a resource has been requested using the full-length URI. Let’s call such a shortened referent a _Temporary Resource Identifier_, _TeRI_ for short. This could be expressed by a response option as shown in Figure 5.

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

Figure 5: Option for offering a TeRI in a response
The TeRI offer option indicates that the server promises to offer this resources under the TeRI given for at least the time given as the duration. Another TeRI offer can be made later to extend the duration.

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

Proposal: Add a TeRI option (e.g., number 2) that can be used in CoAP requests and responses.

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

Do not add any form of stateless URI encoding.

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

As the use of TeRIs is entirely optional, minimal complexity nodes can get by without implementing them.
5. Block-wise transfers

Not all resource representations will fit into a single link layer packet of a constrained network. Using fragmentation (either at the adaptation layer or at the IP layer) to enable the transport of larger representations is possible up to the maximum size of a UDP datagram, but the fragmentation/reassembly process loads the lower layers with conversation state that is better managed in the application layer.

This section proposes options to enable _block-wise_ access to resource representations. The overriding objective is to avoid creating conversation state at the server for block-wise GET requests. (It is impossible to fully avoid creating conversation state for POST/PUT, if the creation/replacement of resources is to be atomic; where that property is not needed, there is no need to create server conversation state in this case, either.) Also, implementation of these options is intended to be optional. (The details of which parts of the behavior need to be mandatory to enable that optionality still are TBD, see below.)

The size of the blocks should not be fixed by the protocol. On the other hand, implementation should be as simple as possible. We therefore propose a small range of power-of-two block sizes, from $2^4$ (16) to $2^{11}$ (2048) bytes. One of these eight values can be encoded in three bits (0 for $2^4$ to 7 for $2^{11}$ bytes), the "szx" (size exponent); the actual block size is then "$1 \ll (szx + 4)$".

5.1. The Block Option

When a representation is larger than can be comfortably transferred in a single UDP datagram, the Block option can be used to indicate a block-wise transfer. Block is a 1-, 2- or 3-byte integer, the four least significant bits of which indicate the size and whether the current block-wise transfer is the last block being transferred (M or "more" bit). The value divided by sixteen is the number of the block currently being transferred, starting from zero, i.e., the current transfer is about the "size" bytes starting at "blocknr $\ll (szx + 4)$". The default value of the Block option is zero, indicating that the current block is the first (block number 0) and only (M bit not set) block of the transfer; however, there is no explicit size implied by this default value.
The block option is used in one of three roles:

- In the request for a GET, it gives the block number requested and suggests a block size (block number 0) or echoes the block size of previous blocks received (block numbers other than 0).

- In the response for a GET or in the request for a PUT or POST, it describes what block number is contained in the payload, and whether further blocks are part of that body (M bit). If the M bit is set, the size of the payload body in bytes MUST indeed be the power of two given by the block size. All blocks for a transaction MUST use the same block size, except for the last block (M bit not set).

- In the response for a PUT or POST, it indicates what block number is being acknowledged. In this case, the M bit is set to indicate that this response does not carry the final response to the request; this can occur when the M bit was set in the request and the server implements PUT/POST atomically (only with the reception of the last block).

In all cases, the block number logically extends the transaction ID, i.e. the same transaction ID can be used for all exchanges for a block-wise transfer. (For GET, and for PUT/POST where atomic
semantics are not needed, the requester is free to use different transactions IDs for each block if desired.)

When a GET is answered with a response carrying a Block option with the M bit set, the requestor may retrieve additional blocks by sending requests with a Block option giving the block number desired. In such a Block option, the M bit MUST be sent as zero and ignored on reception.

To influence the block size used in response to a GET request, the requestor uses the Block option, giving the desired size, a block number of zero and an M bit of zero. A server SHOULD use the block size indicated or a smaller size. Any further block-wise requests for blocks beyond the first one MUST indicate the block size used in the response for the first one.

If the Block option is used by the requestor, all GET requests in a single transaction MUST use the same size. The server SHOULD use the block size indicated in the request option, but the requestor MUST take note of the actual block size used in the response; the server MUST ensure that it uses the same block size for all responses in a transaction (except for the last one with the M bit not set). [TBD: decide whether the Block option can only be used in a response if a Block option was in the request. Such a minimal block option could be of length zero, i.e., would occupy just one byte for the type/length information, but is a bit redundant, so it would be nice to leave this requirement out, but then every GET requestor has the burden of having to cope with receiving Block options.]

Block-wise transfers SHOULD be used in conjunction with the Etag option, unless the representation being exchanged is entirely static (not changing over time at all, such as in a schema describing a device). When reassembling the representation from the blocks being exchanged, the reassembler MUST compare Etag options. If the Etag options do not match in a GET transfer, the requestor 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 transaction, but there is no requirement for the server to establish any state. The server may offer a TeRI with the initial block to reduce the size of further block-wise GET requests; this TeRI MAY be short-lived and specific to the version of the representation being retrieved (which would in effect freeze the representation of the resource specifically for the purposes of this block-wise transfer).

In a PUT or POST transfer, the block option refers to the body in the request, i.e., there is no way to perform a block-wise retrieval of the body of the response. Servers that do need to supply large
bodies in response to PUT/POST SHOULD therefore be employing
redirects, possibly offering a TeRI.

In a PUT or POST transfer that is intended to be implemented in an
atomic fashion at the server, the actual creation/replacement takes
place at the time a block with the M bit unset is received. If not
all previous blocks are available at the server at this time, the
transfer fails and error code 4__ (TBD) MUST be returned. The error
code 4__ can also be returned at any time by a server that does not
currently have the resources to store blocks for a block-wise PUT or
POST transfer that it would intend to implement in an atomic fashion.
[TBD: a way for a server to derive the equivalent of an Etag for the
request body, so that when these do not match in a PUT or POST
transfer, the reassembler MUST discard older blocks. For now, the
transaction ID will have to suffice.]

Proposal: Add a Block option (e.g., number 8) that can be used for
block-wise transfers.

Benefits: Transfers larger than 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
retransmission if required.

Both sides have a say in the block size that actually will be
used.

TBD: Give examples with detailed message flows for a block-wise GET,
PUT and POST.
6. Option Encoding

The option encoding in [I-D.ietf-core-coap] is neither particularly flexible nor particularly efficient. One important, easily overlooked disadvantage of the encoding is the large number of ways in which the same information can be encoded. This unneeded variability causes problems in interoperability and increases both coding and testing efforts required.

6.1. A More Efficient Option Encoding

The basic idea of the proposed encoding is to reduce the number of ways the same information can be encoded as far as possible (but not further). This both simplifies decoding (e.g., an implementation that only ever uses short URIs never has to implement long options, because these can only be used with long lengths) and interoperability testing (there is only one way to say a specific thing, so there aren’t multiple ways that need testing).

One of the undesired variations in packets is the ordering of the options. In this draft, we therefore mandate a total ordering of options, ordered by the option number.

As an interesting consequence, the option numbers can now be expressed in delta coding, in turn requiring fewer bits to encode the option number. This frees a number of bits for the length, making the likelihood of actually needing the two-byte form of the option header much smaller.

To further reduce variation, the length of the value (as always, not including the option header) is now encoded in such a way that there is only one way to express a given length: The short form (one-byte option tag) can express length values from 0 to 14, and the long form is used for values of 15 to 15+255=270, inclusively (Figure 7).

```
+---+---+---+---+---+---+---+---+
| option delta | length     | for 0..14
+---+---+---+---+---+---+---+---+
for 15..270:
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| option delta | 1 1 1 1 | length - 15 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 7: Option delta/length representation with small range

The small option delta of 0..15 in this encoding limits the difference in option value between two adjacent options (or the value
of the option number of the first option). While realistic sequences of options rarely will have a problem here, option numbers 14, 28, ...
are reserved for no-op options with no body (implementations will automatically ignore these with zero additional code; see Section 6.2 why the reserved values are not 15, 30, ...). Note that the resulting delta that reaches the interim nop option may have any number, e.g., for including option 2 and 27 in one message, the sequence would be:

- delta = 2 (option 2, body)
- delta = 12 (option 14 = no-op, no body)
- delta = 13 (option 27, body)

In the unlikely case that only option 40 is needed, the sequence would be:

- delta = 14 (option 14 = no-op, no body)
- delta = 14 (option 28 = no-op, no body)
- delta = 12 (option 40, body)

6.2. Critical Options

CoAP is designed to enable the definition of additional options by later extensions. Typically, new options are designed in such a way that they can simply be ignored if not understood, i.e. new options are _elective_. However, some new options may be _critical_, i.e., there is no good way to process the message if the option is not understood. (Actually, half of the options currently on the table are critical in nature.)

In the option encoding proposed, odd-numbered options indicate a critical option; even-numbered options indicate elective options. (Note that, again, the even/odd distinction is on the option number resulting from the decoding, not the delta value actually embedded in the packet.)

Implementing this proposal requires some renumbering of options from [I-D.ietf-core-coap].

6.3. Errors in Options

When a message contains a critical option that is not understood by the receiver, we say that _decoding fails_.

Bormann & Hartke Expires January 2, 2011 [Page 16]
When a message contains an option that is defined for a specific length of value (e.g., Max-Age, which is only defined for length 1), this is treated like an unknown option. For a critical option, this causes the decoding to fail. For an elective option, this is not an error, the option with the unsupported structure is just ignored. (In both cases, the intention is to allow extension of the option by different syntax in a later revision of the protocol.)

If the decoding of a message fails, the processing depends on the message type:

- NCN messages and RST messages with decode failures are always silently ignored.

- CON messages with decode failures lead to an RST with error code 400 (Bad Request). The payload of the RST SHOULD be a copy of the option bytes that caused decoding to fail. However, nodes with minimal capabilities may choose to restrict their error processing to a minimum.

- ACK messages that fail to decode are hard to process. The requesting node MAY repeat the request with fewer options in order to receive a simpler answer; if that is not possible, the decoding failure should be treated like a client error. Conversely, nodes SHOULD not send critical options in ACK messages unless the CON message eliciting the ACK contained options that justify this. (There may be exceptions, e.g., a node is always allowed to send a Block option with a large resource representation. A requestor that does not understand Block may never be able to receive that resource representation properly, so it is appropriate to treat the situation as a client error.)

6.4. Payload-Length Option

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

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

Problem: The Uri option currently does not provide a way to distinguish an "absolute-URI" from an "absolute-path" [RFC3986], as the leading slash is omitted from the latter. (Ticket #12.)

Proposal: Split the option into two variants: "Uri-Full" and "Uri-Path". None (= "Uri-Path" with option value ''), one of these, but never both can be present.

Problem: The Etag option only allows for up to four bytes in one Etag. If Etags are computed with a random distribution (e.g., by hashing the resource representation), the birthday paradox makes a collision surprisingly likely already for 1e4 variants in circulation.

Proposal: Allow longer Etags (i.e., don’t specify a specific upper limit). The default Apache Etag has about 8..12 Bytes of information in it (file ID = inode number, size, timestamp; which interestingly is mostly redundant with information available in Content-Length and Last-Modified). If a tighter upper limit is desired, 8 Bytes should suffice for all practical purposes, but makes two-way gatewaying with HTTP more complex.
7. IANA Considerations

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

<table>
<thead>
<tr>
<th>Type</th>
<th>C/E</th>
<th>Name</th>
<th>Data type</th>
<th>Length</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>E</td>
<td>TeRI</td>
<td>Duration + Sequence of Bytes</td>
<td>2-n B</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>E</td>
<td>Accept</td>
<td>Sequence of bytes</td>
<td>1-n B</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>C</td>
<td>Block</td>
<td>Unsigned Integer</td>
<td>1-3 B</td>
<td></td>
</tr>
</tbody>
</table>

With the new option encoding and the proposal for essential options, the total list becomes:

<table>
<thead>
<tr>
<th>Type</th>
<th>C/E</th>
<th>Name</th>
<th>Data type</th>
<th>Length</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>E</td>
<td>TeRI</td>
<td>Duration + Sequence of Bytes</td>
<td>2-n B</td>
<td>(none)</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>Uri-Path</td>
<td>String</td>
<td>1-n B</td>
<td>''</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>Accept</td>
<td>Sequence of Bytes</td>
<td>1-n B</td>
<td>any</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>Uri-Full</td>
<td>String</td>
<td>1-n B</td>
<td>(use Uri-Path)</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>Max-age</td>
<td>Duration</td>
<td>1 B</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>C</td>
<td>Content-type</td>
<td>Unsigned Integer</td>
<td>1* B</td>
<td>0 (= text/plain)</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>Etag</td>
<td>Sequence of Bytes</td>
<td>1-4* B</td>
<td>(none)</td>
</tr>
<tr>
<td>8</td>
<td>E</td>
<td>Date</td>
<td>Unsigned Integer</td>
<td>4-6 B</td>
<td>(none)</td>
</tr>
<tr>
<td>Type</td>
<td>C/E</td>
<td>Name</td>
<td>Data type</td>
<td>Length</td>
<td>Rules</td>
</tr>
<tr>
<td>------</td>
<td>-----</td>
<td>-----------------------</td>
<td>-----------</td>
<td>--------</td>
<td>-----------</td>
</tr>
<tr>
<td>13</td>
<td>C</td>
<td>Block</td>
<td>Unsigned Integer</td>
<td>1-3 B</td>
<td>0 (see Section 5.1)</td>
</tr>
<tr>
<td>14.</td>
<td>E</td>
<td>Nop</td>
<td>None</td>
<td>0 B</td>
<td>(‘’)</td>
</tr>
<tr>
<td>15</td>
<td>C</td>
<td>Payload-length</td>
<td>Unsigned Integer</td>
<td>0-2 B</td>
<td>(none)</td>
</tr>
</tbody>
</table>

(The upper limit of "n" indicates that the size is limited only by the options encoding. * indicates that this document proposes to change the limit.) Odd option numbers indicate critical options, even option numbers indicate elective options. Option numbers 14, 28, 42, ... (any number divisible by 14) are reserved (they are elective and therefore ignored by all implementations).

(Subscription-related options are discussed in [I-D.hartke-coap-observe], so the following option from [I-D.ietf-core-coap] is not further discussed here:)

<table>
<thead>
<tr>
<th>Type</th>
<th>C/E</th>
<th>Name</th>
<th>Data type</th>
<th>Length</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>E</td>
<td>Subscription-lifetime</td>
<td>Duration</td>
<td>1 B</td>
<td>With SUBSCRIBE or its response</td>
</tr>
</tbody>
</table>
8. Security Considerations

TBD. (Weigh the security implications of application layer block-wise transfer against those of adaptation-layer or IP-layer fragmentation. Discuss the implications of TeRIs. Also: Discuss nodes without clocks.)

8.1. Amplification Attacks

TBD. (This section discusses how CoAP nodes could become implicated in DoS attacks by using the amplifying properties of the protocol, as well as mitigations for this threat.)
9. References

9.1. Normative References

[I-D.hartke-coap-observe]
Hartke, K. and C. Bormann, "Observing Resources in CoAP", draft-hartke-coap-observe-00 (work in progress), June 2010.

[I-D.ietf-core-coap]

[I-D.ietf-httpbis-p1-messaging]

[I-D.ietf-httpbis-p4-conditional]

[I-D.ietf-httpbis-p6-cache]


9.2. Informative References


Appendix A. Things we won’t do

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

A.1. An efficient stateless URI encoding

There is very little redundancy by repetition in a typical URI, rendering popular compression methods such as LZ77 (as implemented in the widely used DEFLATE algorithm [RFC1951]) rather ineffective.

For the short, non-repetitive data structures that URIs tend to be, efficient stateless compression is pretty much confined to Huffman (or, for even more complexity, arithmetic) coding. The complexity can be reduced significantly by moving to n-ary Huffman coding, i.e., optimizing not to the bit level, but to a larger level of granularity. Informal experiments by the author show that a 16-ary Huffman coding is close to optimal for reasonable URI lengths. In other words, basing the encoding on nibbles (4-bit half-bytes) is both nearly optimal and relatively inexpensive to implement.

The actual letter frequencies that will occur in CoAP URIs are hard to predict. As a stopgap, the author has analyzed an HTTP-based URI corpus and found the following characters to occur with high frequency:

%.aeinorst

In the encoding proposed, each of these ten highly-compressed characters is represented by a single 4-bit nibble. As the % character is used for hexadecimal encoding in URIs, two additional nibbles are used to provide the numeric value of the two hexadecimal numbers following the % character (the original URI will only be properly reconstituted if these are upper-case as they should be according to section 2.1 of the URI specification [RFC3986]; the encoder can choose to send all three characters in dual-nibble format if that matters). An encoder could also map non-ASCII characters to this three-nibble form, even though they are not allowed in URIs. This gives compatibility with the %-encoding required by [RFC3986].

All other characters are represented by both of their nibbles. The resulting sequence of nibbles is reconstituted into a sequence of bytes in most-significant-nibble-first order. Any unused nibble in the last byte of an encoding is set to 0. (Upon decoding, this padding can be readily distinguished from another % combination as this would require another byte after the last byte.) The encoding is summarized in Figure 8.
An example encoding for "/.well-known/resources" (where the initial slash is left out, as proposed for abs-path URIs) is given in Figure 9. While the more than 28% savings in this example may seem just an accident, the HTTP-based corpus indeed shows an average savings of about 21.8%, i.e. the sum of the lengths of the encoded version of all URIs in the corpus is about 78.2% of the sum of the length of all URIs. (The savings should be noticeably higher with a more RESTful selection of URIs than was available for this experiment.)

Figure 9: Nibble-based URI encoding: 21 -> 15 bytes
Appendix B. Experimental Options

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

B.1. Options indicating absolute time

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

- "Date", defined in Section 14.18 in [RFC2616] (Section 9.3 in [I-D.ietf-httpbis-p1-messaging]), giving the absolute time a response was generated;

- "Last-Modified", defined in Section 14.29 in [RFC2616] (Section 6.6 in [I-D.ietf-httpbis-p4-conditional], giving the absolute time of when the origin server believes the resource representation was last modified;

- "If-Modified-Since", defined in Section 14.25 in [RFC2616], "If-Unmodified-Since", defined in Section 14.28 in [RFC2616], and "If-Range", defined in Section 14.27 in [RFC2616] can be used to supply absolute time to gate a conditional request;

- "Expires", defined in Section 14.21 in [RFC2616] (Section 3.3 in [I-D.ietf-httpbis-p6-cache]), giving the absolute time after which a response is considered stale.

- The more obscure headers "Retry-After", defined in Section 14.37 in [RFC2616], and "Warning", defined in section 14.46 in [RFC2616], also may employ absolute time.

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

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

In UNIX-related systems, it is customary to indicate absolute time as an integer number of seconds, after midnight UTC, January 1, 1970. Unless negative numbers are employed, this time format cannot represent time values prior to January 1, 1970, which probably is not required for the uses of absolute time in CoAP.
If a 32-bit integer is used and allowance is made for a sign-bit in a local implementation, the latest UTC time value that can be represented by the resulting 31 bit integer value is 03:14:07 on January 19, 2038. If the 32-bit integer is used as an unsigned value, the last date is 2106-02-07, 06:28:15.

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

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

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

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

Carsten Bormann
Universitaet Bremen T2I
Postfach 330440
Bremen  D-28359
Germany

Phone: +49-421-218-63921
Fax:   +49-421-218-7000
Email: cabo@tzi.org

Klaus Hartke
Universitaet Bremen T2I
Postfach 330440
Bremen  D-28359
Germany

Phone: +49-421-218-63905
Fax:   +49-421-218-7000
Email: hartke@tzi.org