core
Internet-Draft
Intended status: Standards Track
Expires: August 7, 2015

P. van der Stok
consultant
B. Greevenbosch
independent
A. Bierman
YumaWorks
J. Schoenwaelder
A. Sehgal
Jacobs University
February 3, 2015

CoAP Management Interface
draft-vanderstok-core-comi-06

Abstract

This document describes a network management interface for constrained devices, called CoMI. CoMI is an adaptation of the RESTCONF protocol for use in constrained devices and networks. It is designed to reduce the message sizes, server code size, and application development complexity. The Constrained Application Protocol (CoAP) is used to access management data resources specified in YANG, or SMIv2 converted to YANG. The payload of the CoMI message is encoded in Concise Binary Object Representation (CBOR).

Note

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

Status of This Memo

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy and building control. Constrained devices need to be managed in an automatic fashion to handle the large quantities of devices that are expected in future installations. The messages between devices need to be as small and infrequent as possible. The implementation complexity and runtime resources need to be as small as possible.

The draft [I-D.ietf-netconf-restconf] describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG [RFC6020]. RESTCONF allows access to data resources contained in NETCONF [RFC6241] datastores. RESTCONF messages can be encoded in XML [XML] or JSON. The GET method is used to retrieve data resources and the POST, PUT, PATCH, and DELETE methods are used to create, replace, merge, and delete data resources.

A large amount of Management Information Base (MIB) [RFC3418] specifications already exist for monitoring purposes. This data can be accessed in RESTCONF if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

The CoRE Management Interface (CoMI) is intended to work on standardized data-sets in a stateless client-server fashion. The RESTCONF protocol is adapted and optimized for use in constrained environments, using CoAP instead of HTTP. Standardized data sets promote interoperability between small devices and applications from different manufacturers. Stateless communication is encouraged to keep communications simple and the amount of state information small in line with the design objectives of 6lowpan [RFC4944] [RFC6775], RPL [RFC6650], and CoAP [RFC7252].

RESTCONF uses the HTTP methods HEAD, OPTIONS, and PATCH, which are not available in CoAP. HTTP uses TCP which is not recommended for CoAP. The transport protocols available to CoAP are much better suited for constrained networks.

TODO: Introduce CoAP Patch options to allow modification to subsets of resource.

CoMI is low resource oriented, uses CoAP, and only supports the methods GET, PUT, POST and DELETE. The payload of CoMI is encoded in CBOR [RFC7049] which is automatically generated from JSON [JSON]. CBOR has a binary format and hence has more coding efficiency than JSON. To promote small packets, CoMI uses an additional data identifier string to number conversion to minimise CBOR payloads and URI length. It is assumed that the managed device is the most constrained entity. The client might be more capable, however this is not necessarily the case.

Currently, small managed devices need to support at least two protocols: CoAP and SNMP. When the MIB can be accessed with the CoAP protocol, the SNMP protocol can be replaced with the CoAP protocol. Although the SNMP server size is not huge (see Appendix A), the code for the security aspects of SMIv3 is not negligible. Using CoAP to access secured management objects reduces the code complexity of the stack in the constrained device, and harmonizes applications development.

The objective of CoMI is to provide a CoAP based Function Set that reads and sets values of managed objects in devices to (1) initialize parameter values at start-up, (2) acquire statistics during operation, and (3) maintain nodes by adjusting parameter values during operation.

The end goal of CoMI is to provide information exchange over the CoAP transport protocol in a uniform manner as a first step to the full management functionality as specified in [I-D.ersue-constrained-mgmt].

1.1. Design considerations

CoMI supports discovery of resources, accompanied by reading, writing and notification of resource values. As such it is close to the device management of the Open Mobile Alliance described in [OMA]. A detailed comparison between CoMI and LWM2M management can be found in Appendix C. CoMI supports MIB modules which have been translated from SMIv2 to YANG, using [RFC6643]. This mapping is read-only so writable SMIv2 objects need to be converted to YANG using an implementation-specific mapping.

CoMI uses a simple URI to access the management object resources. Complexity introduced by instance selection, or multiple object specification is expressed with uri-query attributes. The choice for uri-query attributes makes the URI structure less context dependent.

TODO: Use of YANG data model reduces message size.

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].

Readers of this specification should be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

The following terms are defined in the NETCONF protocol [RFC6241]: client, configuration data, datastore, and server.

The following terms are defined in the YANG data modelling language [RFC6020]: container, data node, key, key leaf, leaf, leaf-list, and list.

The following terms are defined in RESTCONF protocol [I-D.ietf-netconf-restconf]: data resource, datastore resource, edit operation, query parameter, target resource, and unified datastore.

The following terms are defined in this document:

YANG hash: CoMI object identifier, which is a 30-bit numeric hash of the YANG object identifier string for the object. When a YANG hash value is printed in a request target URI, error-path or other string, then the lowercase hexadecimal representation is used. Leading zeros are used so the value uses 8 hex characters.

The following list contains the abbreviations used in this document.

XXXX: TODO, and others to follow.

1.2.1. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

Brackets "[" and "]" enclose list keys.

Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

2. CoMI Architecture

This section describes the CoMI architecture to use CoAP for the reading and modifying of instrumentation variables used for the management of the instrumented node.

```
   Client
   +--------------------------------------------------------------+
   | +----------------+    +----------------+                     |
   | |    SMIv2       | >  |      YANG      |    >     COAP       |
   | |specification(2)|    |specification(1)|       Request(3)    |
   | +----------------+    +----------------+[         *          |
   +-----------------------------*-----------[---------*----------+
                                 *           [         *
                                 *           [    +-----------+
                         mapping *   security[    |  Network  |
                                 *      (8)  [    | packet(4) |
                                 *           [    +-----------+
   Server                        *           [         *
   +-----------------------------*-----------[---------*----------+
   |                             *           [         *          |
   |                             *                 Retrieval,     |
   |                             *               Modification(5)  |
   |                            \*/                    *          |
   | +-------------------------------------------------*--------+ |
   | |                    +--------------+       +------------+ | |
   | |                    |configuration |       |Operational | | |
   | |                    |     (6b)     |       |  state(6a) | | |
   | |                    +--------------+       +------------+ | |
   | |                    variable store (6)           *        | |
   | +-------------------------------------------------*--------+ |
   |                                                   *          |
   |                                                Variable      |
   |                                            Instrumentation(7)|
   +--------------------------------------------------------------+
```

Figure 1: Abstract CoMI architecture

Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

Objectives are:

- o Equip a constrained node with a management server that provides information about the operational characteristics of the code running in the constrained node.
- o The server provides this information in a variable store that contains values describing the performance characteristics and the code parameter values.
- o The client receives the performance characteristics on a regular basis or on request.
- o The client sets the parameter values in the server at bootstrap and intermittently when operational conditions change.
- o The constrained network requires the payload to be as small as possible, and the constrained server memory requirements should be as small as possible.

For interoperability it is required that in addition to using the Internet Protocol for data transport:

- o The names, type, and semantics of the instrumentation variables are standardized.
- o The instrumentation variables are described in a standard language.
- o The signature of the CoAP request in the server is standardized.
- o The format of the packet payload is standardized.
- o The notification from server to client is standardized.

The different numbered components of Figure 1 are discussed according to component number.

(1) YANG specification: contains a set of named and versioned modules. A module specifies a hierarchy of named and typed resources. A resource is uniquely identified by a sequence of its name and the names of the enveloping resources following the hierarchy order. The YANG specification serves as input to the writers of application and instrumentation code and the humans

analysing the returned values (arrow from YANG specification to Variable store). The specification can be used to check the correctness of the CoAP request and do the CBOR encoding.

(2) SMIv2 specification: A named module specifies a set of variables and "conceptual tables". Named variables have simple types. Conceptual tables are composed of typed named columns. The variable name and module name identify the variable uniquely. There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoAP request: The CoAP request needs a Universal Resource Identifier (URI) and the payload of the packet to send a request. The URI is composed of the schema, server, path and query and looks like coap://entry.example.com/<path>?<query>. Fragments are not supported. Allowed operations are PUT, GET, DELETE, and POST. New variables can be created with POST when they exist in the YANG specification. The Observe option can be used to return variable values regularly or on event occurrence (notification).

(3.1) CoAP <path>: The path identifies the variable in the form "/mg/<hash-value>".

(3.2) CoAP <query>: The query parameter is used to specify additional (optional) aspects like the module name, the smi context, and others. The idea is to keep the path simple and put variations on variable specification in the query.

(3.3) CoAP discovery: Discovery of the variables is done with standard CoAP resource discovery using /.well-known/core with ?rt=/core.mg.

(4) Network packet: The payload contains the CBOR encoding of JSON objects. This object corresponds to the converted RESTCONF message payload.

(5) Retrieval, modification: The server needs to parse the CBOR encoded message and identify the corresponding instances in the Variable store. In addition, this component includes the code for CoAP Observe and block options.

(6) Variable store: The store is composed of two parts: Operational state and Configuration datastore (see Section 2.1). CoMI does not see the different variable store types. The Variable store contains instances of the YANG specification. Values are stored in the appropriate instances, and or values are returned from the instances into the payload of the packet.

(7) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects. The Variable instrumentation code stores the values of the parameters into the appropriate places in the operational code. The variable instrumentation code reads current execution values from the operational code and stores them in the appropriate instances.

(8) Security: The server MUST prevent unauthorized users from reading or writing any data resources. CoMI relies on DTLS which is specified to secure CoAP communication.

2.1. RESTCONF/YANG Architecture

CoMI adapts the RESTCONF architecture so data exchange and implementation requirements are optimized for constrained devices.

The RESTCONF protocol uses a unified datastore to edit conceptual data structures supported by the server. The details of transaction preparation and non-volatile storage of the data are hidden from the RESTCONF client. CoMI also uses a unified datastore, to allow stateless editing of configuration variables and the notification of operational variables.

The child schema nodes of the unified datastore include all the top-level YANG data nodes in all the YANG modules supported by the server. The YANG data structures represent a hierarchy of data resources. The client discovers the list of YANG modules, and important conformance information such as the module revision dates, YANG features supported, and YANG deviations required. The individual data nodes are discovered indirectly by parsing the YANG modules supported by the server.

The YANG data definition statements contain a lot of information that can help automation tools, developers, and operators use the data model correctly and efficiently. The YANG definitions and server YANG module capability advertisements provide an "API contract" that allow a client to determine the detailed server management capabilities very quickly. CoMI allows access to the same data resources as a RESTCONF server, except the messages are optimized to reduce identifier and payload size.

RESTCONF uses a simple algorithmic mapping from YANG to URI syntax to identify the target resource of a retrieval or edit operation. A client can construct operations or scripts using a predictable syntax, based on the YANG data definitions. The target resource URI can reference a data resource instance, or the datastore itself (to retrieve the entire datastore or create a top-level data resource instance). CoMI uses a 30-bit YANG hash value (based on the YANG data node path identifier strings) to identify schema nodes in the target resource URI and in the payload.

Any message payload data is relative to the node specified in the target resource URI in a request message. CoMI message payloads are based on the JSON encoding of a RESTCONF message payload. The JSON identifier names are first converted to their 30-bit YANG hash values and then the payload is converted to CBOR.

3. CoAP Interface

In CoAP a group of links can constitute a Function Set. The format of the links is specified in [I-D.ietf-core-interfaces]. This note specifies a Management Function Set. CoMI end-points that implement the CoMI management protocol support at least one discoverable management resource of resource type (rt): core.mg, with path: /mg, where mg is short-hand for management. The name /mg is recommended but not compulsory (see Section 4.4).

The mg resource has three sub-resources accessible with the paths:

/mg: YANG-based data with path "/mg" and using CBOR content encoding format. This path represents a datastore resource which contains YANG data resources as its descendant nodes. All identifiers referring to YANG data nodes within this path are encoded as YANG hash values (see Section 5.4.

/mg/mod.uri: URI indicating the location of the server module information, with path "/mg/mod.uri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG hash values.

/mg/yang.hash: URI indicating the location of the server YANG hash information if any objects needed to be re-hashed by the server. It has path "/mg/yang.hash" and is encoded in CBOR format. The "ietf-yang-hash" module of Section 5.3 is used to define the syntax and semantics of this data structure. This YANG data is encoded with plain identifier strings, not YANG hash values. The server will only have this resource if there are any objects that needed to be re-hashed due to a hash collision.

The mapping of YANG data nodes to CoMI resources is as follows: A YANG module describes a set of data trees composed of YANG data nodes. Every root of a data tree in a YANG module loaded in the CoMI server represents a resource of the server. All data root descendants represent sub-resources.

The resource identifiers of the instances of the YANG specifications are YANG hash values, as described in Section 5.1. When multiple instances of a list node exist, the instance selection is described in Section 4.1.3.4

The profile of the management function set, with IF=core.mg, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

name	path	rt	Data Type
Management	/mg	core.mg	n/a
Data	/mg	core.mg.data	application/cbor
Module Set URI	/mg/mod.uri	core.mg.moduri	application/cbor
YANG Hash Info	/mg/yang.hash	core.mg.yang-hash	application/cbor

4. MG Function Set

The MG Function Set provides a CoAP interface to perform a subset of the functions provided by RESTCONF.

A subset of the operations defined in RESTCONF are used in CoMI:

Operation	Description
GET	Retrieve the datastore resource or a data resource
POST	Create a data resource
PUT	Create or replace a data resource
DELETE	Delete a data resource

4.1. Data Retrieval

4.1.1. GET

One or more instances of data resources are retrieved by the client with the GET method. The RESTCONF GET operation is supported in CoMI. The same constraints apply as defined in section 3.3 of [I-D.ietf-netconf-restconf]. The operation is mapped to the GET method defined in section 5.8.1 of [RFC7252].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [I-D.ietf-core-block] is used. Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

There are two query parameters for the GET method. A CoMI server MUST implement the keys parameter and MAY implement the select parameter to allow common data retrieval filtering functionality.

Query Parameter	Description
keys	Request to select instances of a YANG definition
select	Request selected sub-trees from the target resource

The "keys" parameter is used to specify a specific instance of the resource. When keys is not specified, all instances are returned. When no or one instance of the resource exists, the keys parameter is not needed.

4.1.2. Mapping of the 'select' Parameter

ANUJ TODO: Add more details based on the RESTCONF 'select' parameter. We need to add information about how this parameter is encoded. There should there be an error notification when filtering fails.

RESTCONF uses the 'select' parameter to specify an expression which can represent a subset of all data nodes within the target resource [I-D.ietf-netconf-restconf]. This parameter is useful for filtering sub-trees and retrieving only a subset that a managing application is interested in.

However, filtering is a resource intensive task and not all constrained devices can be expected to have enough computing resources such that they will be able to successfully filter and return a subset of a sub-tree. This is especially likely to be true with Class 0 devices that have significantly lesser RAM than 10 KiB [RFC7228]. Since CoMI is targeted at constrained devices and networks, only a limited subset of the ’select’ parameter is used here.

Unlike the RESTCONF ’select’ parameter, CoMI does not use object names in "XPath" or "path-expr" format to identify the subset that needs to be filtered. Parsing XML is resource intensive for constrained devices [management] and using object names can lead to large message sizes. Instead, CoMI utilizes the YANG hashes described in Section 5 to identify the sub-trees that should be filtered from a target resource. Using these hashes ensures that a constrained node can identify the target sub-tree without expending many resources and that the messages generated are also efficiently encoded.

The implementation of the ’select’ parameter is already optional for constrained devices, however, even when implemented it is expected to be a best effort feature, rather than a service that nodes must provide. This implies that if a node receives the ’select’ parameter specifying a set of sub-trees that should be returned, it will only return those that it is able to.

4.1.3. Retrieval Examples

The examples in this section use a JSON payload with one or more entries describing the pair (identifier, value). CoMI transports the CBOR format to transport the equivalent contents. The CBOR syntax of the payloads is specified in Section 5.

4.1.3.1. Single instance retrieval

A request to read the values of instances of a management object or the leaf of an object is sent with a confirmable CoAP GET message. A single object is specified in the URI path prefixed with /mg.

Using for example the clock container from [RFC7317], a request is sent to retrieve the value of clock/current-datetime specified in module system-state. The answer to the request returns a (identifier, value) pair.

In all examples: (1) the payload is expressed in JSON, although the operational payload is specified to be in CBOR, (2) the path is expressed in readable names although the transported path is expressed a hash value of the name (where the hash value in the payload is expressed as a hexadecimal number, and the hash value in the URL as a baseb 64 number), and (3) only one instance is associated with the resource.

```
REQ: GET example.com/mg/system-state/clock/current-datetime

RES: 2.05 Content (Content-Format: application/cbor)
{
    "current-datetime" : "2014-10-26T12:16:31Z"
}
```

The YANG hash value for ’current-datetime’ is calculated by constructing the schema node identifier for the object:

```
/sys:system-state/sys:clock/sys:current-datetime
```

The 30 bit murmur3 hash value is calculated on this string (0x15370408 and VNwQI). The request using this hash value is shown below:

```
REQ: GET example.com/mg/VNwQI

RES: 2.05 Content (Content-Format: application/cbor)
{
    0x15370408 : "2014-10-26T12:16:31Z"
}
```

The specified object can be an entire object. Accordingly, the returned payload is composed of all the leaves associated with the object. Each leaf is returned as a (YANG hash, value) pair. For example, the GET of the clock object, sent by the client, results in the following returned payload sent by the managed entity:

```
REQ: GET example.com/mg/system-state/clock
   (Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
   "clock" : {
      "current-datetime" : "2014-10-26T12:16:51Z",
      "boot-datetime" : "2014-10-21T03:00:00Z"
   }
}
```

The YANG hash values for ’clock’, ’current-datetime’, and ’boot-datetime’ are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):

```
/sys:system-state/sys:clock (0x2eb2fa3b and usvo7)
/sys:system-state/sys:clock/sys:current-datetime (0x15370408)
/sys:system-state/sys:clock/sys:boot-datetime (0x1fa25361)
```

The request using the hash values is shown below:

```
REQ: GET example.com/mg/usvo7
   (Content-Format: application/cbor)

RES: 2.05 Content (Content-Format: application/cbor)
{
   0x2eb2fa3b : {
      0x15370408 : "2014-10-26T12:16:51Z",
      0x1fa25361 : "2014-10-21T03:00:00Z"
   }
}
```

4.1.3.2. Multiple instance retrieval

The specified list node can have multiple instances. Accordingly, the returned payload is composed of all the instances associated with the list node. Each instance is returned as a (identifier, value) pair. For example, the GET of the /interfaces/interface/ipv6/neighbor instance identified with interface index "eth0" [RFC7223], sent by the client, results in the following returned payload sent by the managed entity:

```
   REQ: GET example.com/mg/interfaces/interface/ipv6/neighbor?keys=eth0
      (Content-Format: application/cbor)

   RES: 2.05 Content (Content-Format: application/cbor)
   {
      "neighbor" : [
        {
       "ip" : "fe80::200:f8ff:fe21:67cf",
       "link-layer-address" : "00:00::10:01:23:45"
        },
        {
       "ip" : "fe80::200:f8ff:fe21:6708",
       "link-layer-address" : "00:00::10:54:32:10"
        },
        {
       "ip" : "fe80::200:f8ff:fe21:88ee",
       "link-layer-address" : "00:00::10:98:76:54"
        }
      ]
   }
```

The YANG hash values for ’neighbor’, ’ip’, and ’link-layer-address’ are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):

```
/if:interfaces/if:interface/ip:ipv6/ip:neighbor (0x2354bc49 and jVLxJ)
/if:interfaces/if:interface/ip:ipv6/ip:neighbor/ip:ip (0x20b8907e and guJB_)
/if:interfaces/if:interface/ip:ipv6/ip:neighbor/ip:link-layer-address
   (0x16f47fd8)
```

The request using the hash values is shown below:

```
   REQ: GET example.com/mg/jVLxJ?keys=eth0
      (Content-Format: application/cbor)

   RES: 2.05 Content (Content-Format: application/cbor)
   {
      0x2354bc49 : [
        {
       0x20b8907e : "fe80::200:f8ff:fe21:67cf",
       0x16f47fd8 : "00:00::10:01:23:45"
        },
        {
       0x20b8907e : "fe80::200:f8ff:fe21:6708",
       0x16f47fd8 : "00:00::10:54:32:10"
        },
        {
       0x20b8907e : "fe80::200:f8ff:fe21:88ee",
       0x16f47fd8 : "00:00::10:98:76:54"
        }
      ]
   }
```

4.1.3.3. Access to MIB Data

The YANG translation of the SMI specifying the ipNetToMediaTable yields:

```
   container IP-MIB {
     container ipNetToPhysicalTable {
       list ipNetToPhysicalEntry {
          key "ipNetToPhysicalIfIndex ipNetToPhysicalNetAddressType
               ipNetToPhysicalNetAddress";
          leaf ipNetToMediaIfIndex {
             type: int32;
          }
          leaf ipNetToPhysicalIfIndex {
            type if-mib:InterfaceIndex;
          }
          leaf ipNetToPhysicalNetAddressType {
            type inet-address:InetAddressType;
          }
          leaf ipNetToPhysicalNetAddress {
            type inet-address:InetAddress;
          }
          leaf ipNetToPhysicalPhysAddress {
            type yang:phys-address {
               length "0..65535";
            }
          }
          leaf ipNetToPhysicalLastUpdated {
            type yang:timestamp;
          }
          leaf ipNetToPhysicalType {
            type enumeration { ... }
          }
          leaf ipNetToPhysicalState {
            type enumeration { ... }
          }
          leaf ipNetToPhysicalRowStatus {
            type snmpv2-tc:RowStatus;
          }
       }
    }
```

The following example shows an "ipNetToPhysicalTable" with 2 instances, using JSON encoding:

```
{
  "IP-MIB" {
    "ipNetToPhysicalTable" : {
      "ipNetToPhysicalEntry" : [
        {
          "ipNetToPhysicalIfIndex" : 1,
          "ipNetToPhysicalNetAddressType" : "ipv4",
          "ipNetToPhysicalNetAddress" : "10.0.0.51",
          "ipNetToPhysicalPhysAddress" : "00:00:10:01:23:45",
          "ipNetToPhysicalLastUpdated" : "2333943",
          "ipNetToPhysicalType" : "static",
          "ipNetToPhysicalState" : "reachable",
          "ipNetToPhysicalRowStatus" : "active"
        },
        {
          "ipNetToPhysicalIfIndex" : 1,
          "ipNetToPhysicalNetAddressType" : "ipv4",
          "ipNetToPhysicalNetAddress" : "9.2.3.4",
          "ipNetToPhysicalPhysAddress" : "00:00:10:54:32:10",
          "ipNetToPhysicalLastUpdated" : "2329836",
          "ipNetToPhysicalType" : "dynamic",
          "ipNetToPhysicalState" : "unknown",
          "ipNetToPhysicalRowStatus" : "active"
        }
      ]
    }
  }
}
```

The YANG hash values for ’ipNetToPhysicalEntry’ and its child nodes are calculated by constructing the schema node identifier for the objects, and then calculating the 30 bit murmur3 hash values (shown in parenthesis):

```
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable (0x30b7bc3f and wt7w_)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry
    (0x1067f289 and QZ/KJ)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry/
    ip-mib:ipNetToPhysicalIfIndex (0x00d38564)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry/
    ip-mib:ipNetToPhysicalNetAddressType (0x2745e222)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry/
    ip-mib:ipNetToPhysicalNetAddress (0x387804eb)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry/
    ip-mib:ipNetToPhysicalPhysAddress (0x1a51514a)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry/
    ip-mib:ipNetToPhysicalLastUpdated (0x03f95578)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry/
    ip-mib:ipNetToPhysicalType (0x24ade115)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry/
    ip-mib:ipNetToPhysicalState (0x09e640ef)
 /ip-mib:IP-MIB/ip-mib:ipNetToPhysicalTable/ip-mib:ipNetToPhysicalEntry/
    ip-mib:ipNetToPhysicalRowStatus (0x3b5c1ab6)
```

The following example shows a request for the entire ipNetToPhysicalTable. Since all the instances are requested, no "keys" query parameter is needed.

```
   REQ: GET example.com/mg/IP-MIB/ipNetToPhysicalTable

   REQ: GET example.com/mg/wt7w_

   RES: 2.05 Content (Content-Format: application/cbor)
   {
      0x30b7bc3f : {
         0x1067f289 : [
           {
             0x00d38564 : 1,
             0x2745e222 : "ipv4",
             0x387804eb : "10.0.0.51",
             0x1a51514a : "00:00:10:01:23:45",
             0x03f95578 : "2333943",
             0x24ade115 : "static",
             0x09e640ef : "reachable",
             0x3b5c1ab6 : "active"
           },
           {
             0x00d38564 : 1,
             0x2745e222 : "ipv4",
             0x387804eb : "9.2.3.4",
             0x1a51514a : "00:00:10:54:32:10",
             0x03f95578 : "2329836",
             0x24ade115 : "dynamic",
             0x09e640ef : "unknown",
             0x3b5c1ab6 : "active"
           }
         ]
      }
   }
```

4.1.3.4. The ’keys’ Query Parameter

There is a mandatory query parameter that MUST be supported by servers called "keys". This parameter is used to specify the key values for an instance of an object identified by a YANG hash value. Any key leaf values of the instance are passed in order. The first key leaf in the top-most list is the first key encoded in the ’keys’ parameter.

The key leafs from top to bottom and left to right are encoded as a comma-delimited list. If a key leaf value is missing then all values for that key leaf are returned.

Example: In this example exactly 1 instance is requested from the ipNetToPhysicalEntry (from a previous example).

```
   REQ: GET example.com/mg/IP-MIB/ipNetToPhysicalTable/
      ipNetToPhysicalEntry?keys=1,ipv4,10.0.0.51

   REQ: GET example.com/mg/QZ/KJ?keys=1,ipv4,10.0.0.51

   RES: 2.05 Content (Content-Format: application/cbor)
   {
      0x1067f289 : [
         {
           0x00d38564 : 1,
           0x2745e222 : "ipv4",
           0x387804eb : "10.0.0.51",
           0x1a51514a : "00:00:10:01:23:45",
           0x03f95578 : "2333943",
           0x24ade115 : "static",
           0x09e640ef : "reachable",
           0x3b5c1ab6 : "active"
         }
      ]
   }
```

An example illustrates the syntax of keys query parameter. In this example the following YANG module is used:

```
     module foo-mod {
       namespace foo-mod-ns;
       prefix foo;

       list A {
         key "key1 key2";
         leaf key1 { type string; }
         leaf key2 { type int32; }
         list B {
           key "key3";
           leaf key3 { type string; }
           leaf col1 { type uint32; }
         }
       }
     }
```

The path identifier for the leaf "col1" is the following string:

```
      /foo:A/foo:B/foo:col1
```

The YANG has value for this identifier string 0xa9abdcca and pq9zK).

The following string represents the RESTCONF target resource URI expression for the "col1" leaf for the key values "top", 17, and "group1":

```
   /restconf/data/foo-mod:A=top,17/B=group1/col1
```

The following string represents the CoMI target resource identifier for the same instance of the "col1" leaf:

```
   /mg/pq9zK?keys=top,17,group1
```

4.2. Data Editing

CoMI allows datastore contents to be created, modified and deleted using CoAP methods.

TODO: Data-editing is an optional feature. A server can choose to only support YANG modules with read-only objects.

4.2.1. POST

Data resource instances are created with the POST method. The RESTCONF POST operation is supported in CoMI, however it is only allowed for creation of data resources. The same constraints apply as defined in section 3.4.1 of [I-D.ietf-netconf-restconf]. The operation is mapped to the POST method defined in section 5.8.2 of [RFC7252].

There are no query parameters for the POST method.

TODO: CoMI does not support user-ordered lists in YANG.

4.2.2. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. A request to set the values of instances of an object/leaf is sent with a confirmable CoAP PUT message. The Response is piggybacked to the CoAP ACK message corresponding with the Request. The same constraints apply as defined in section 3.5 of [I-D.ietf-netconf-restconf]. The operation is mapped to the PUT method defined in section 5.8.3 of [RFC7252].

There are no query parameters for the PUT method.

TODO: Define where PATCH is needed.

4.2.3. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI. The same constraints apply as defined in section 3.7 of [I-D.ietf-netconf-restconf]. The operation is mapped to the DELETE method defined in section 5.8.4 of [RFC7252].

There are no optional query parameters for the PUT method.

4.3. Notify functions

Notification by the server to a selection of clients when the value of a management object changes is an essential function for the management of servers. CoMI allows to do a notifications on all variables in the datastore.

Notification of object changes is supported with the CoAP Observe [I-D.ietf-core-observe] function. The client subscribes to the object by sending a GET request with an "Observe" option.

```
REQ: GET example.com/mg/ietf-ip/ipv6/neighbor/ip
    (observe option register)

RES: 2.05 Content (Content-Format: application/cbor)
{
    "ip" : "fe80::200:f8ff:fe21:67cf"
}
```

The same example with the hash values instead of the string identifiers looks like:

```
REQ: GET example.com/mg/guJB_
    (observe option register)

RES: 2.05 Content (Content-Format: application/cbor)
{
    0x20b8907e : "fe80::200:f8ff:fe21:67cf"
}
```

In the example, the request returns a success response with the contents of the ip field. Consecutively the server will regularly notify the client when ip changes value.

To check that the client is still alive, the server MUST send confirmable notifications once in a while. When the client does not confirm the notification from the server, the server will remove the client from the list of observers[I-D.ietf-core-observe].

In the registration request, the client MAY include a "Response-To-Uri-Host" and optionally "Response-To-Uri-Port" option as defined in [I-D.becker-core-coap-sms-gprs]. In this case, the observations SHOULD be sent to the address and port indicated in these options. This can be useful when the client wants the managed device to send the trap information to a multicast address.

4.4. Module Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.mg" [RFC6690]. Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, but it is recommended that the value "/mg" is used, where possible. The example below shows the discovery of the presence and location of management data.

```
REQ: GET /.well-known/core?rt=core.mg

RES: 2.05 Content </mg>; rt="core.mg"
```

Management objects MAY be discovered with the standard CoAP resource discovery. The implementation can add the hash values of the object identifiers to /.well-known/core with rt="core.mg.data". The available objects identified by the hash values can be discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.mg.data". Upon success, the return payload will contain the registered hash values and their location. The example below shows the discovery of the presence and location of management data.

```
     REQ: GET /.well-known/core?rt=core.mg.data

     RES: 2.05 Content </mg/BaAiN>; rt="core.mg.data",
   </mg/CF_fA>; rt="core.mg.data"; obs
```

In the example the "obs" attribute indicates that the object /mg/ CF_fA is observed.

Lists of hash values may become prohibitively long. It is discouraged to provide long lists of objects on discovery. Therefore, it is recommended that details about management objects are discovered following the RESTCONF protocol. The YANG module information is stored in the "ietf-yang-library" module [I-D.ietf-netconf-restconf]. The resource "/mg/mod.uri" is used to retrieve the location of the YANG module library.

Since many constrained servers within a deployment are likely to be similar, the module list can be stored locally on each server, or remotely on a different server.

```
     Local in example.com server:

     REQ: GET example.com/mg/mod.uri

     RES: 2.05 Content (Content-Format: application/cbor)
     {
       "mod.uri" : "example.com/mg/modules"
     }


     Remote in example-remote-server:

     REQ: GET example.com/mg/mod.uri

     RES: 2.05 Content (Content-Format: application/cbor)
     {
       "moduri" : "example-remote-server.com/mg/group17/modules"
     }
```

Within the YANG module library all information about the module is stored such as: module identifier, identifier hierarchy, grouping, features and revision numbers.

The hash identifier is obtained as specified in Section 5.1. When a collision occurred in the name space of the target server, a rehash is executed.

4.5. Error Return Codes

The RESTCONF return status codes defined in section 6 of the RESTCONF draft are used in CoMI error responses, except they are converted to CoAP error codes.

TODO: complete RESTCONF to CoAP error code mappings

RESTCONF Status Line	CoAP Status Code
100 Continue	none?
200 OK	2.05
201 Created	2.01
202 Accepted	none?
204 No Content	?
304 Not Modified	2.03
400 Bad Request	4.00
403 Forbidden	4.03
404 Not Found	4.04
405 Method Not Allowed	4.05
409 Conflict	none?
412 Precondition Failed	4.12
413 Request Entity Too Large	4.13
414 Request-URI Too Large	4.00
415 Unsupported Media Type	4.15
500 Internal Server Error	5.00
501 Not Implemented	5.01
503 Service Unavailable	5.03

5. Mapping YANG to CoMI payload

A mapping for the encoding of YANG data in CBOR is necessary for the efficient transport of management data in the CoAP payload. Since object names may be rather long and may occur repeatedly, CoMI allows for association of a given object path identifier string value with an integer, called a "YANG hash".

5.1. YANG Hash Generation

The association between string value and string number is done through a hash algorithm. The 30 least significant bits of the "murmur3" 32-bit hash algorithm are used. This hash algorithm is described online at http://en.wikipedia.org/wiki/MurmurHash. Implementation are available online, including at https://code.google.com/p/smhasher/wiki/MurmurHash. When converting 4 input bytes to a 32-bit integer in the hash algorithm, the Little-Endian convention MUST be used.

The hash is generated for the string representing the object path identifier. A canonical representation of the path identifier is used.

> Prefix values are used on every node.
>
> The prefix values defined in the YANG module containing the data object are used for the path expression. For external modules, this is the value of the ’prefix’ sub-statement in the ’import’ statement for each external module.
>
> Path expressions for objects which augment data nodes in external modules are calculated in the augmenting module, using the prefix values in the augmenting module.
>
> Choice and case node names are not included in the path expression. Only ’container’, ’list’, ’leaf’, ’leaf-list’, and ’anyxml’ nodes are listed in the path expression.

The "murmur3_32" hash function is executed for the entire path string. The value ’42’ is used as the seed for the hash function. The YANG hash is subsequently calculated by taking the 30 least significant bits.

The resulting 30-bit number is used by the server, unless the value is already being used for a different object by the server. In this case, the re-hash procedure in the following section is executed.

5.2. Re-Hash Procedure

A hash collision occurs if two different path identifier strings have the same hash value. If the server has over 38,000 objects in its YANG modules, then the probability of a collision is fairly high. If a hash collision occurs on the server, then the object that is causing the conflict has to be altered, such that the new hash value does not conflict with any value already in use by the server.

In most cases, the hash function is expected to produce unique values for all the objects supported by a constrained device. Given a known set of YANG modules, both server and client can calculate the YANG hashes independently, and offline.

Even though collisions are expected to happen rather rarely, they needs to be considered. Collisions can be detected before deployment, if the vendor knows which modules are supported by the server, and hence all YANG hashes can be calculated. Collisions are only an issue when they occur at the same server. The client needs to discover any re-hash mappings on a per server basis.

If the server needs to re-hash any object identifiers, then it MUST create a "rehash-map" entry for the altered identifier, as described in the following YANG module.

5.3. ietf-yang-hash YANG Module

The "ietf-yang-hash" YANG module is used by the server to report any objects that have been mapped to produce a new hash value that does not conflict with any other YANG hash values used by the server.

YANG tree diagram for "ietf-yang-hash" module:

```
   +--ro yang-hash
      +--ro rehash* [hash]
         +--ro hash      uint32
         +--ro path?     string
         +--ro append?   string
```

```
module ietf-yang-hash {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-hash";
  prefix "yh";

  organization
    "IETF CORE (Constrained RESTful Environments) Working Group";

  contact
    "WG Web:   <http://tools.ietf.org/wg/core/>
     WG List:  <mailto:core@ietf.org>

     WG Chair: Carsten Bormann
               <mailto:cabo@tzi.org>

     WG Chair: Andrew McGregor
```

```
                <mailto:andrewmcgr@google.com>

      Editor:   Peter van der Stok
                <mailto:consultancy@vanderstok.org>

      Editor:   Bert Greevenbosch
                <mailto:andy@bert.greevenbosch@huawei.com>

      Editor:   Andy Bierman
                <mailto:andy@yumaworks.com>

      Editor:   Juergen Schoenwaelder
                <mailto:j.schoenwaelder@jacobs-university.de>

      Editor:   Anuj Sehgal
                <mailto:s.anuj@jacobs-university.de>";

     description
       "This module contains re-hash information for the CoMI protocol.

        Copyright (c) 2014 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Simplified BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (http://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX; see
        the RFC itself for full legal notices.";

     // RFC Ed.: replace XXXX with actual RFC number and remove this
     // note.

     // RFC Ed.: remove this note
     // Note: extracted from draft-vanderstok-core-comi-05.txt

     // RFC Ed.: update the date below with the date of RFC publication
     // and remove this note.
     revision 2014-10-27 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: CoMI Protocol.";
     }
     container yang-hash {
       config false;
       description
         "Contains information on the YANG Hash values used by
          the server.";

       list rehash {
         key hash;
         description
           "Each entry describes an re-hash mapping in use by
            the server.";

         leaf hash {
           type uint32;
           description "The hash value that has a collision";
         }
         leaf path {
           type string;
           description
             "The YANG identifier path expression that caused the
              collision and is being remapped";
         }
         leaf append {
           type string;
           description
             "The string that the server appended to the path
              expression contained in the 'path' leaf to produce
              a new path expression and therefore new hash value.
              The YANG hash value for the new string (identified
              by 'path' + 'append') is used to identify the
              'path' object.";
         }
       }
     }

   }
```

5.3.1. YANG Re-Hash Example

In this example the server has an object that is already registered when the "/foo:A/foo:B/foo:col1" object is processed. This object path string hashes to value 0x29abdcca. The server has appended the string "_" to the path to produce a new hash (0x2a7a2044) which does not collide with any other objects.

The server would return the following information if the client retrieved the "/mg/yang-hash" resource.

```
REQ: GET example.com/mg/yang-hash

RES: 2.05 Content (Content-Format: application/cbor)
{
   "ietf-yang-hash:yang-hash" : {
      "rehash" : [
         {
            "hash" : 712646724,
            "path" :"/foo:A/foo:B/foo:col1",
            "append" : "_"
         }
      ]
   }
}
```

5.4. YANG Hash in URL

When a URL contains a YANG hash, it is encoded using base64url "URL and Filename safe" encoding as specified in [RFC4648].

The hash H is represented as a 30-bit integer, divided into five 6-bit integers as follows:

```
B1 = (H & 0x3f000000) >> 24
B2 = (H & 0xfc0000) >> 18
B3 = (H & 0x03f000) >> 12
B4 = (H & 0x000fc0) >> 6
B5 =  H & 0x00003f
```

Subsequently, each 6-bit integer Bx is translated into a character Cx using Table 2 from [RFC4648], and a string is formed by concatenating the characters in the order C1, C2, C3, C4, C5.

For example, the YANG hash 0x29abdcca is encoded as "pq9zK".

6. Mapping YANG to CBOR

6.1. High level encoding

When encoding YANG variables in CBOR, the CBOR encodings entry is a map. The key is the YANG hash of entry variable, whereas the value contains its value.

For encoding of the variable values, a CBOR datatype is used. Section 6.2 provides the mapping between YANG datatypes and CBOR datatypes.

6.2. Conversion from YANG datatypes to CBOR datatypes

Table 1 defines the mapping between YANG datatypes and CBOR datatypes.

Elements of types not in this table, and of which the type cannot be inferred from a type in this table, are ignored in the CBOR encoding by default. Examples include the "description" and "key" elements. However, conversion rules for some elements to CBOR MAY be defined elsewhere.

YANG type	CBOR type	Specification
int8, int16, int32, int64, uint16, uint32, uint64, decimal64	unsigned int (major type 0) or negative int (mayor type 1)	The CBOR integer type depends on the sign of the actual value.
boolean	either "true" (major type 7, simple value 21) or "false" (major type 7, simple value 20)	
string	text string (major type 3)	
enumeration	unsigned int (major type 0)	
bits	array of text strings	Each text string contains the name of a bit value that is set.
binary	byte string (major type 2)	
empty	null (major type 7, simple value 22)	TBD: This MAY not be applicable to true MIBs, as SNMP may not support empty variables...
union		Similar ot the JSON transcription from

		[I-D.ietf-netmod-yang-json], the elements in a union MUST be determined using the procedure specified in section 9.12 of [RFC6020].
leaf-list	array (major type 4)	The array is encapsulated in the map associated with the YANG variable.
list	array (major type 4) of maps (major type 5)	Each array element contains a map of associated YANG hash - value pairs.
container	map (major type 5)	The map contains YANG hash - value pairs corresponding to the elements in the container.
smiv2:oid	array of integers	Each integer contains an element of the OID, the first integer in the array corresponds to the most left element in the OID.

Table 1: Conversion of YANG datatypes to CBOR

7. Error Handling

In case a request is received which cannot be processed properly, the managed entity MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code, and SHOULD include additional information in the payload.

Such an error message payload is encoded in CBOR, using the following structure:

TODO: Adapt RESTCONF <errors> data structure for use in CoMI. Need to select the most important fields like <error-path>.

```
errorMsg     : ErrorMsg;

*ErrorMsg {
  errorCode  : uint;
  ?errorText : tstr;
}
```

The variable "errorCode" has one of the values from the table below, and the OPTIONAL "errorText" field contains a human readable explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.00	General error
1	4.00	Malformed CBOR data
2	4.00	Incorrect CBOR datatype
3	4.00	Unknown MIB variable
4	4.00	Unknown conversion table
5	4.05	Attempt to write read-only variable
0..2	5.01	Access exceptions
0..18	5.00	SMI error status

The CoAP error code 5.01 is associated with the exceptions defined in [RFC3416] and CoAP error code 5.00 is associated with the error-status defined in [RFC3416].

8. Security Considerations

For secure network management, it is important to restrict access to MIB variables only to authorised parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. This includes DTLS for protected access to resources, as well suitable authentication and authorisation mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

In addition, mechanisms for authentication and authorisation may need to be selected.

CoMI avoids defining new security mechanisms as much as possible. However some adaptations may still be required, to cater for CoMI’s specific requirements.

9. IANA Considerations

’rt="core.mg.data"’ needs registration with IANA.

’rt="core.mg.moduri"’ needs registration with IANA.

’rt="core.mg.yang-hash"’ needs registration with IANA.

Content types to be registered:

- o application/comi+cbor

10. Acknowledgements

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. Carsten Bormann has given feedback on the use of CBOR. The draft has benefited from comments (alphabetical order) by Dee Denteneer, Esko Dijk, Michael van Hartskamp, Zach Shelby, Michel Veillette, Michael Verschoor, and Thomas Watteyne. The CBOR encoding borrows extensively from Ladislav Lhotka’s description on conversion from YANG to JSON.

11. Changelog

Changes from version 00 to version 01

- o Focus on MIB only

- o Introduced CBOR, JSON, removed BER

- o defined mappings from SMI to xx

- o Introduced the concept of addressable table rows

Changes from version 01 to version 02

- o Focus on CBOR, used JSON for examples, removed XML and EXI

- o added uri-query attributes mod and con to specify modules and contexts

- Definition of CBOR string conversion tables for data reduction
- use of Block for multiple fragments
- Error returns generalized
- SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- Added security considerations

Changes from version 03 to version 04

- Added design considerations section
- Extended comparison of management protocols in introduction
- Added automatic generation of CBOR tables
- Moved lowpan table to Appendix

Changes from version 04 to version 05

- Merged SNMP access with RESTCONF access to management objects in small devices
- Added CoMI architecture section
- Added RESTCONf NETMOD description
- Rewrote section 5 with YANG examples
- Added server and payload size appendix
- Removed Appendix C for now. It will be replaced with a YANG example.

Changes from version 04 to version 05

- Extended examples with hash representation
- Added keys query parameter text
- Added select query parameter text
- Better separation between specification and instance

- o Section on discovery updated
- o Text on rehashing introduced
- o Elaborated SMI MIB example
- o Yang libary use described
- o use of BigEndian/LittleEndian in Hash generation specified

Changes from version 05 to version 06

- o Hash values in payload as hexadecimal and in URL in base64 numbers
- o Streamlined CoMI architecture text
- o Added select query parameter text
- o Data editing optional
- o Text on Notify added
- o Text on rehashing improved with example

12. References

12.1. Normative References

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[I-D.becker-core-coap-sms-gprs]
Becker, M., Li, K., Kuladinithi, K., and T. Poetsch, "Transport of CoAP over SMS", draft-becker-core-coap-sms-gprs-05 (work in progress), August 2014.

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", draft-ietf-core-block-16 (work in progress), October 2014.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-16 (work in progress), December 2014.

[I-D.ietf-netmod-yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", draft-ietf-netmod-yang-json-02 (work in progress), November 2014.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-04 (work in progress), January 2015.

12.2. Informative References

[RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets:MIB-II", STD 17, RFC 1213, March 1991.

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, December 2002.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, December 2002.

[RFC3416] Presuhn, R., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.

[RFC3418] Presuhn, R., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, December 2002.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4088] Black, D., McCloghrie, K., and J. Schoenwaelder, "Uniform Resource Identifier (URI) Scheme for the Simple Network Management Protocol (SNMP)", RFC 4088, June 2005.

[RFC4113] Fenner, B. and J. Flick, "Management Information Base for the User Datagram Protocol (UDP)", RFC 4113, June 2005.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[RFC4293] Routhier, S., "Management Information Base for the Internet Protocol (IP)", RFC 4293, April 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.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

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

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, July 2012.

[RFC6650] Falk, J. and M. Kucherawy, "Creation and Use of Email Feedback Reports: An Applicability Statement for the Abuse Reporting Format (ARF)", RFC 6650, June 2012.

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

[RFC6775] Shelby, Z., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, November 2012.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, August 2014.

[RFC7388] Schoenwaelder, J., Sehgal, A., Tsou, T., and C. Zhou, "Definition of Managed Objects for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7388, October 2014.

[RFC7390] Rahman, A. and E. Dijk, "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, October 2014.

[I-D.ietf-core-interfaces]
Shelby, Z. and M. Vial, "CoRE Interfaces", draft-ietf-core-interfaces-02 (work in progress), November 2014.

[I-D.ersue-constrained-mgmt]
Ersue, M., Romascanu, D., and J. Schoenwaelder, "Management of Networks with Constrained Devices: Problem Statement, Use Cases and Requirements", draft-ersue-constrained-mgmt-03 (work in progress), February 2013.

[I-D.ietf-lwig-coap]
Kovatsch, M., Bergmann, O., Dijk, E., He, X., and C. Bormann, "CoAP Implementation Guidance", draft-ietf-lwig-coap-01 (work in progress), July 2014.

[STD0001] "Official Internet Protocols Standard", Web http://www.rfc-editor.org/rfcxx00.html, .

[XML] "Extensible Markup Language (XML)", Web http://www.w3.org/xml, .

[JSON] "JavaScript Object Notation (JSON)", Web http://www.json.org, .

[OMA]   "OMA-TS-LightweightM2M-V1_0-20131210-C", Web http://technical.openmobilealliance.org/Technical/current_releases.aspx, .

[DTLS-size]
Hummen, R., Shafagh, H., Raza, S., Voigt, T., and K. Wehrle, "Delegation-based Authentication and Authorization for the IP-based Internet of Things", Web http://www.vs.inf.ethz.ch/publ/papers/mshafagh_secon14.pdf, .

[dcaf]   Bormann, C., Bergmann, O., and S. Gerdes, "Delegated Authenticated Authorization for Constrained Environments", Private Information , .

[openwsn] Watteijne, T., "Coap size in Openwsn", Web http://builder.openwsn.org/, .

[Erbium]   Kovatsch, M., "Erbium Memory footprint for coap-18", Private Communication , .

[management]
Schoenwalder, J. and A. Sehgal, "Management of the Internet of Things", Web http://cnds.eecs.jacobs-university.de/slides/2013-im-iot-management.pdf, 2013.

Appendix A. Payload and Server sizes

This section provides information on code sizes and payload sizes for a set of management servers. Approximate code sizes are:

Code	processor	Text	Data	reference
Observe agent	erbium	800	n/a	[Erbium]
CoAP server	MSP430	1K	6	[openwsn]
SNMP server	ATmega128	9K	700	[management]
Secure SNMP	ATmega128	30K	1.5K	[management]
DTLS server	ATmega128	37K	2K	[management]
NETCONF	ATmega128	23K	627	[management]
JSON parser	CC2538	4.6K	8	[dcaf]
CBOR parser	CC2538	1.5K	2.6K	[dcaf]
DTLS server	ARM7	15K	4	[I-D.ietf-lwig-coap]
DTLS server	MSP430	15K	4	[DTLS-size]
Certificate	MSP430	23K		[DTLS-size]
Crypto	MSP430	2-8K		[DTLS-size]

Thomas says that the size of the CoAP server is rather arbitrary, as its size depends mostly on the implementation of the underlying library modules and interfaces.

Payload sizes are compared for the following request payloads, where each attribute value is null (N.B. these sizes are educated guesses, will be replaced with generated data). The identifier are assumed to be a string representation of the OID. Sizes for SysUpTime differ due to preambles of payload. "CBOR opt" stands for CBOR payload where the strings are replaced by table numbers.

Request	BERR SNMP	JSON	CBOR	CBOR opt
IPnetTOMediaTable	205	327	˜327	˜51
lowpanIfStatsTable		710	614	121
sysUpTime	29	13	˜13	20
RESTconf example				

Appendix B. Notational Convention for CBOR data

To express CBOR structures [RFC7049], this document uses the following conventions:

A declaration of a CBOR variable has the form:

```
name : datatype;
```

where "name" is the name of the variable, and "datatype" its CBOR datatype.

The name of the variable has no encoding in the CBOR data.

"datatype" can be a CBOR primitive such as:

tstr: A text string (major type 3)

uint: An unsigned integer (major type 0)

map(x,y): A map (major type 5), where each first element of a pair is of datatype x, and each second element of datatype y. A '.' character for either x or y means that all datatypes for that element are valid.

A datatype can also be a CBOR structure, in which case the variable's "datatype" field contains the name of the CBOR structure. Such CBOR structure is defined by a character sequence consisting of first its name, then a '{' character, then its subfields and finally a '}' character.

A CBOR structure can be encapsulated in an array, in which case its name in its definition is preceded by a '*' character. Otherwise the structure is just a grouping of fields, but without actual encoding of such grouping.

The name of an optional field is preceded by a '?' character. This means, that the field may be omitted if not required.

Appendix C. comparison with LWM2M

TODO: Anuj promised text

Authors' Addresses

Peter van der Stok
consultant

Phone: +31-492474673 (Netherlands), +33-966015248 (France)
Email: consultancy@vanderstok.org
URI: www.vanderstok.org

Bert Greevenbosch
independent

Email: ietf@bertgreevenbosch.nl

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Juergen Schoenwaelder
Jacobs University
Campus Ring 1
Bremen 28759
Germany

Email: j.schoenwaelder@jacobs-university.de

Anuj Sehgal
Jacobs University
Campus Ring 1
Bremen 28759
Germany

Email: s.anuj@jacobs-university.de