core P. van der Stok
Internet-Draft consultant
Intended status: Standards Track A. Bierman
Expires: September 8, 2016 YumaWorks
March 7, 2016

CoAP Management Interface
draft-vanderstok-core-comi-09

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. The Constrained Application Protocol (CoAP) is used to access management data resources specified in YANG, or SMIv2 converted to YANG. CoMI use the YANG to CBOR mapping and encodes YANG names to reduce payload size.

Note

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

Status of This Memo

Copyright Notice

Table of Contents

1. 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.

This draft describes the CoAP Management Interface which uses CoAP methods to access structured data defined in YANG [RFC6020]. This draft is complementary to the draft [I-D.ietf-netconf-restconf] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data sets, specified in a standardized language such as YANG, promotes interoperability between devices and applications from different manufacturers. A large amount of Management Information Base (MIB) [RFC3418] [mibreg] specifications already exists for monitoring purposes. This data can be accessed in RESTCONF or CoMI if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

RESTCONF allows access to data resources contained in NETCONF [RFC6241] data-stores. RESTCONF messages can be encoded in XML [XML] or JSON [RFC7159]. 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.

CoMI supports the methods GET, PUT, PATCH, POST and DELETE. The payload of CoMI is encoded in CBOR [RFC7049] which can be automatically generated from JSON [RFC7159]. CBOR has a binary format and hence has more coding efficiency than JSON. RESTCONF relies on HTTP with TCP in contrast to CoMI which uses CoAP that is optimized for UDP with less overhead for small messages. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

To promote small packets, CoMI uses an additional "data-identifier string-to-number conversion" to minimize CBOR payloads and URI length.

Currently, small managed devices need to support at least two protocols: CoAP and SNMP [RFC3411]. When the MIB can be accessed with the CoMI 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 [RFC3414] 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.

1.1. Terminology

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

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, data-store, 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, data-store resource, edit operation, query parameter, target resource, and unified data-store. The terms YANG hash, and Rehash bit are defined in I-D.yang-hash.

The following terms are defined in this document:

Data-node instance: An instance of a data-node specified in a YANG module present in the server. The instance is stored in the memory of the server.

Notification-node instance: An instance of a schema node of type notification, specified in a YANG module present in the server. The instance is generated in the server at the occurrence of the corresponding event and appended to a stream.

The following list contains the abbreviations used in this document.

XXXX: TODO, and others to follow.

1.1.1. Tree Diagrams

A simplified graphical representation of the data model is used in the YANG modules specified 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:

- Equip a constrained node with a management server that provides information about the operational characteristics of the code running in the constrained node.

- The server provides this information in a variable store that contains values describing the performance characteristics and the code parameter values.

- The client receives the performance characteristics on a regular basis or on request.

- The client sets the parameter values in the server at bootstrap and intermittently when operational conditions change.

- 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:

- The names, type, and semantics of the instrumentation variables are standardized.

- The instrumentation variables are described in a standard language.

- The URI of the CoAP request is standardized.

- The format of the packet payload is standardized.

- 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 analyzing 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, PATCH, GET, DELETE, and POST. New variables can be created with POST when they exist in the YANG specification. The Observe option is used to return variable values regularly or on event occurrence (notification).

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

(3.2) CoAP <query>: The query parameter is used to specify additional (optional) aspects like the container name, list instance, 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 with 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 data-store (see Section 2.1). CoMI does not differentiate between variable store types. The Variable store contains data-node instances. 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 [RFC6347] 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 data-store 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 data-store, to allow stateless editing of configuration variables and the notification of operational variables.

The child schema nodes of the unified data-store 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

RESTCONF and CoMI use 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 data-store itself (to retrieve the entire data-store or create a top-level data resource instance). A compression algorithm reduces the size of the data-node instance identifier (see Section 2.2).

2.1.1. Major differences between RESTCONF and CoMI

CoMI uses CoAP/UDP as transport protocol and CBOR as payload format. RESTCONF uses HTTP/TCP as transport protocol and JSON or XML as payload formats. CoMI encodes YANG name strings as numbers, where RESTCONF does not.

CoAP servers MUST maintain the order of user-ordered data. CoMI does not support insert-mode (first, last, before, after) and insertion-point (before, after) which are supported by RESTCONF. Many CoAP servers will not support date and time functions. For that reason CoMI does not support the start, stop options for events.

The CoMI "select" query parameter is equivalent to the RESTCONF "fields" query parameter but has a much simpler syntax. CoMI servers only implement the efficient "trim" mode for default values. CoMI servers implement a less rich syntax to specify key values in the URI than RESTCONF servers.

CoMI servers do not support ’filter’ query that involves XML parsing, ’content’, ’depth’, and ’with-defaults’ query parameters.

CoMI servers do not support the YANG functionality of anyxml, anydata, and xpath.

2.2. Compression of data-node instance identifier

The JSON/CBOR encoding will include the module name string to specify the YANG module. If a representation of the target resource is included in the request or response message, then the data definition name string is used to identify each node in the message. The module namespace (or name) may also be present in these identifiers.

In order to significantly reduce the size of identifiers used in CoMI, numeric object identifiers are used instead of these strings. The specific encoding of the object identifiers is not hard-wired in the protocol.

YANG Hash is the default encoding for object identifiers [I-D.bierman-core-yang-hash]. This encoding in considered to be "unstructured" since the particular values for each object are determined by a hash algorithm. It is possible for 2 different objects to generate the same hash value. If this occurs, then the client and server will both need to rehash the colliding object identifiers to new unused hash values.

In order to eliminate the need for rehashing, CoMI allows for alternate "structured" object identifier encoding formats. Structured object identifier MUST be managed such that no object ID collisions are possible, and therefore no rehash procedures are needed. Structured object identifiers can also be selected to minimize the size of a subset of the object identifiers (e.g., the most requested objects).

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 4.6).

The path prefix /mg has resources accessible with the following five paths:

/mg: YANG-based data with path "/mg" and using CBOR content encoding format. This path represents a data-store resource which contains YANG data resources as its descendant nodes. All identifiers referring to YANG data nodes within this path are encoded YANG names (see for example [I-D.bierman-core-yang-hash].

/mg/mod.uri: URI identifying 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 encoded values. An Entity Tag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

/mg/num.typ: String identifying the object ID numbering scheme used by the CoMI server. Two values are defined in this document: (1) 'yanghash' to indicate that the YANG Hash numbering scheme is used, and (2) 'yangmanag' to indicate that a managed numbering scheme is used. It is possible for other object numbering schemes to be defined outside the scope of this document.

/mg/srv.typ: String identifying the CoMI server type. The value 'ro' indicates that the server is a read-only server and no editing operations are supported. A read-only server is not required to provide YANG deviation statements for any writable YANG data nodes. The value 'rw' indicates that the server is a read-write server and editing operations are supported. A read-write server is required to provide YANG deviation statements for any writable YANG data nodes that are not fully implemented.

/mg/yh.uri: URI indicating the location of the server YANG hash information if any objects needed to be re-hashed by the server. It has the path "/mg/yh.uri" and is encoded in CBOR format. The "yang-hash" container within the "ietf-yang-hash" module, described in [I-D.bierman-core-yang-hash], 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.

/mg/stream: String identifying the default stream resource to which YANG notification instances are appended. Notification support is optional, so this resource will not exist if the server does not support any notifications.

/mg/op: String identifying the resource to which YANG operations are appended.

The mapping of YANG data node instances 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 encoded YANG names. When multiple instances of a list node exist, the instance selection is described in Section 4.1.4.4

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
Numbering Type	/mg/num.typ	core.mg.num-type	application/cbor
Server Type	/mg/srv.typ	core.mg.srv-type	application/cbor
YANG Hash Info	/mg/yh.uri	core.mg.yang-hash	application/cbor
Events	/mg/stream	core.mg.stream	application/cbor
Operations	/mg/op	core.mg.op	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 data-store resource or a data resource
POST	Create a data resource
PUT	Create or replace a data resource
PATCH	Replace a data resource partially
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, as explained in more detail in Section 4.5.

There are three query parameters for the GET method. A CoMI server MUST implement the keys parameter and the content 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 to select sub-trees from the target resource
content	Request to select configuration and non-configuration nodes

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

4.1.2. Using the ’select’ Parameter

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

Because 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, the ’filter’ parameter is not used here.

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 return.

4.1.3. Using the ’content’ query parameter

The "content" parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
config	Return only configuration descendant data nodes
nonconfig	Return only non-configuration descendant data nodes
all	Return all descendant data nodes

This parameter is only allowed for GET methods on datastore and data resources. A 4.00 Bad Request error is returned if used for other methods or resource types.

The default value is determined by the "config" statement value of the requested data nodes. If the "config" value is "false", then the default for the "content" parameter is "nonconfig". If "config" is "true" then the default for the "content" parameter is "config".

4.1.4. Retrieval Examples

In all examples the path is expressed in readable names and as a encoded value of the name (where the encoded value in the payload is expressed as a hexadecimal number, and the encoded value in the URL as a base64 number). CoMI payloads use the CBOR format. The CBOR syntax of the YANG payloads is specified in TODO REF. The examples in this section use a JSON payload with extensions to approach the permissible CBOR payload, called "diagnostic JSON".

4.1.4.1. 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, transported as a CBOR map with a single item.

```
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 specified object can be an entire object. Accordingly, the returned payload is composed of all the leaves associated with the object. The payload is a CBOR map where each leaf is returned as a (encoded YANG name, 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, transported as A CBOR map with two items:

```
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"
  }
}
```

For example, the YANG names can be replaced by the hash values for ’clock’, ’current-datetime’, and ’boot-datetime’. The 30 bit murmur3 hash value of ’clock’ equates: CDKSQ (xref target="I-D.bierman-core-yang-hash"/>. The request using hash values is shown below:

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

RES: 2.05 Content (Content-Format: application/cbor)
{
  0x021ca491 : {

      0x047c468b : "2014-10-26T12:16:51Z",
      0x1fb5f4f8 : "2014-10-21T03:00:00Z"
   }
}
```

4.1.4.2. Multiple instance retrieval

A "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 (key, object) pair, where key and object are composed of one or more (identifier, value) pairs.

For example, the GET of the /interfaces/interface/ipv6/neighbor results in the following returned payload sent by the managed entity, transported as a CBOR map of 3 (key : object) pairs, where key and value are CBOR maps with one entry each. In this case the key is the "ip" attribute and the value is the "link-layer-address" attribute.

```
REQ: GET example.com/mg/interfaces/interface/ipv6/neighbor
   (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"}
   }
}
```

4.1.4.3. Access to MIB Data

The YANG translation of the SMI specifying the ipNetToMediaTable [RFC4293] 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 as defined in [I-D.ietf-netmod-yang-json]:

```
   {
     "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"
           }
         ]
       }
     }
   }
```

4.1.4.4. The ’keys’ Query Parameter

There is a 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 an encoded YANG name. All 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 one instance is requested from the ipNetToPhysicalEntry (from a previous example). The CBOR payload, here represented with diagnostic JSON, permits to transport the selected instance and nothing more.

TODO refer to the section in YANG to CBOR mapping

```
REQ: GET example.com/mg/IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry?key
s=1,ipv4,9.2.3.4

RES: 2.05 Content (Content-Format: application/cbor)
{
   "IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry": {
      {
          "ipNetToPhysicalIfIndex" : 1,
          "ipNetToPhysicalNetAddressType" : "ipv4",
          "ipNetToPhysicalNetAddress" : "9.2.3.4"}:
        {
          "ipNetToPhysicalPhysAddress" : "00:00:10:54:32:10",
          "ipNetToPhysicalLastUpdated" : "2329836",
          "ipNetToPhysicalType" : "dynamic",
          "ipNetToPhysicalState" : "unknown",
          "ipNetToPhysicalRowStatus" : "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 following string represents the CoMI target resource identifier for the instance of the "col1" leaf with key values "top", 17, "group":

```
      /mg/foo-mod:A/B/col1?keys="top",17,"group1"
```

4.1.4.5. The ’select’ Query Parameter

The select parameter is used along with the GET method to provide a sub-tree filter mechanism. A list of encoded YANG names that should be filtered is provided along with a list of keys identifying the instances that should be returned. When the keys parameter is used together with the select, the key values are added in brackets without using the "keys=" text.

Data may be retrieved using the select query parameter in the following way, transported as a CBOR maps of maps:

```
REQ: GET example.com/mg/IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry?sel
ect=(10.0.0.51)

RES: 2.05 Content (Content-Format: application/cbor)
{
    "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"
        }
   }
}
```

In this example exactly one instance is returned as response from the ipNetToPhysicalTable because only this instance matches the provided keys.

Supposing there were multiple YANG fields with their own sets of keys that were to be filtered, the select query parameter can be used to retrieve results from these in one go as well. Using the "foo-mod" module of Section 4.1.4.4, the following string represents the CoMI target resource identifier when multiple fields, with their own sets of key values are queried:

```
       /mg/foo-mod:A?select=B/col1("top",17,"group"),key2("top")
```

4.1.4.6. Defaults handling

If the target of a GET method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST not return the leaf.

If the target of a GET method is a data node that represents a container or list that has any child resources with default values, for the child resources that have not been given value yet, the server MUST not return the child resource.

4.2. Data Editing

CoMI allows data-store contents to be created, modified and deleted using CoAP methods.

Data-editing is an optional feature. The server will indicate its editing capability with the "/core.mg.srv-type resource type. If the value is ’rw’ then the server supports editing operations. If the value is ’ro’ then the server does not support editing operations.

4.2.1. Data Ordering

A CoMI server is not required to support entry insertion of lists and leaf-lists that are ordered by the user (i.e., YANG statement "ordered-by user"). The ’insert’ and ’point’ query parameters from RESTCONF are not used in CoMI.

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as arrays so messages will preserve their order.

4.2.2. POST

Data resource instances are created with the POST method. The RESTCONF POST operation is supported in CoMI for creation of data resources and the invocation operation resources. Refer to Section 4.4 for details on operation resources. The same constraints apply as defined in section 4.4.1 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.

4.2.3. 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.

4.2.4. PATCH

Data resource instances are partially replaced with the PATCH method [I-D.vanderstok-core-patch]. The PATCH operation is supported in CoMI. A request to set the values of instances of a subset of the values of the resource is sent with a confirmable CoAP PATCH 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 PATCH method defined in [I-D.vanderstok-core-patch].

The processing of the PATCH command is specified by the CBOR payload. The CBOR patch payload describes the changes to be made to target YANG data nodes. It follows closely the rules described in [RFC7396]. If the CBOR patch payload contains objects that are not present in the target, these objects are added. If the target contains the specified object, the contents of the objects are replaced with the values of the payload. Null values indicate the removal of existing values. The CBOR patch extends [RFC7396] by specifying rules for list elements.

TODO, review text after publication of YANG/CBOR and CBOR-merge drafts.

For example consider the following YANG specification:

```
   module foo {
     namespace "http://example.com/book";
     prefix "bo";
     revision 2015-06-07;

     list B {
         key key1;
         key key2;
         leaf key1 { type string; }
         leaf key2 {type string; }
         leaf col1 { type int32; }
         leaf counter1 { type uint32; }
     }

     container book {
       leaf title { type string; }
       container author {
         leaf  givenName {type string; }
         leaf  familyName {type string; }
       }
       leaf-list tags {type string; }
       leaf content{type string;}
       leaf phoneNumber {type string;}
     }
```

Consider the following target data nodes described with the JSON encoding of [I-D.ietf-netmod-yang-json].

```
 "B": [
     {
      "key1" : "author1",
      "key2" : "book2",
      "col1" : 25,
      "counter1" : 4321
     },
    {
      "key1" : "author5",
      "key2" : "book6",
      "col1" : 2,
      "counter1" : 1234
     }
  ]


 "book": {
  "title" : "mytitle",
  "author": {
    "givenName" : "John",
    "familyName" : "Doe"
    }
  "tags" : [ "example", "sample"],
  "content" : "This will be unchanged"
}
```

The following changes are requested for the document (following the example from [RFC7396]: the title changes from "mytitle" to "favoured", the phoneNumber is added to the book container, the familyName is deleted, and "sample" is removed from the tags leaf-list. In addition author1, book1 item is removed, author5 counter1 is upgraded, and a new author is added in B list. The following CBOR Patch payload, represented in JSON is sent.

TODO: edit after publication of CBOR-merge draft.

```
 {
    "B":  {
     { "key1" : "author1",
      "key2" : "book2"}:
      { null : null},
     { "key1" : "author5"} :
       {"counter1" : 4444},
     { "key1" : "newauthor",
      "key2" : "newbook"}:
     { "col1" : 1,
       "counter1" : 1}
    },
  "book" : {
     "title" : "favoured",
     "author": {"familyName" : null},
     "tags" : [ "example"],
     "phoneNumber" : "+01-123-456-7890"
   }
}
```

In his example, the value "author5" specifies the entry uniquely. However, when several entries exist with the "author5" value for "key1", the outcome of the example Patch is undefined.

The processing of the Patch payload results in the following new target data nodes.

```
    "B": [
        {
         "key1" : "newauthor",
         "key2" : "newbook",
         "col1" : 1,
         "counter1" : 1
        },
       {
         "key1" : "author5",
         "key2" : "book6",
         "col1" : 2,
         "counter1" : 4444
        }
     ]

   "book": {
     "title" : "favoured",
     "author": {
       "givenName" : "John"
       }
     "tags" : [ "example"],
     "content" : "This will be unchanged",
     "phoneNumber" : +01-123-456-7890"
   }
```

There are no query parameters for the PATCH method.

4.2.5. 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 DELETE method.

4.2.6. Editing Multiple Resources

Editing multiple data resources at once can allow a client to use fewer messages to make a configuration change. It also allows multiple edits to all be applied or none applied, which is not possible if the data resources are edited one at a time.

It is easy to add multiple entries at once. The "PATCH" method can be used to simply patch the parent node(s) of the data resources to be added. If multiple top-level data resources need to be added, then the data-store itself ('/mg') can be patched.

If other operations need to be performed, or multiple operations need to be performed at once, then the YANG Patch [I-D.ietf-netconf-yang-patch] media type can be used with the PATCH method. A YANG patch is an ordered list of edits on the target resource, which can be a specific data node instance, or the data-store itself. The resource type used by YANG Patch is 'application/yang.patch'. A status message is returned in the response, using resource type 'application/yang.patch.status'.

The following YANG tree diagram describes the YANG Patch structure, Each 'edit' list entry has its own operation, sub-resource target, and new value (if needed).

```
  +--rw yang-patch
      +--rw patch-id?   string
      +--rw comment?    string
      +--rw edit* [edit-id]
         +--rw edit-id      string
         +--rw operation    enumeration
         +--rw target       target-resource-offset
         +--rw point?       target-resource-offset
         +--rw where?       enumeration
         +--rw value
```

Refer to [I-D.ietf-netconf-yang-patch] for more details on the YANG Patch request and response contents.

4.3. Notify functions

Notification by the server to a selection of clients when an event occurs in the server is an essential function for the management of servers. CoMI allows events specified in YANG [RFC5277] to be notified to a selection of requesting clients. The server appends newly generated events to a stream. There is one, so-called "default", stream in a CoMI server. The /mg/stream resource identifies the default stream. The server MAY create additional streams. When a CoMI server generates an internal event, it is appended to the chosen stream, and the contents of a notification instance is ready to be sent to all CoMI clients which observe the chosen stream resource.

Reception of generated notification instances is enabled with the CoAP Observe [I-D.ietf-core-observe] function. The client subscribes to the notifications by sending a GET request with an "Observe" option, specifying the /mg/stream resource when the default stream is selected.

Every time an event is generated, the chosen stream is cleared, and the generated notification instance is appended to the chosen stream. After appending the instance, the contents of the instance is sent to all clients observing the modified stream.

Suppose the server generates the event specified with:

```
module example-port {
  ...
  prefix ep;
  ...
  notification example-port-fault {
    description
      "Event generated if a hardware fault on a
       line card port is detected";
    leaf port-name {
      type string;
      description "Port name";
    }
    leaf port-fault {
      type string;
      description "Error condition detected";
    }
  }
}

}
```

By executing a GET on the /mg/stream resource the client receives the following response:

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

RES: 2.05 Content (Content-Format: application/cbor)
{
      "example-port-fault":{
            "port-name" : "0/4/21",
            "port-fault" : "Open pin 2"
      }
}
```

In the example, the request returns a success response with the contents of the last generated event. Consecutively the server will regularly notify the client when a new event is generated.

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. RPC statements

An operation resource represents a protocol operation defined with the YANG "rpc" statement. It is invoked using a POST method on the operation resource with a Token value as specified in section 5.3 "Request/Resonse matching" of [RFC7252] to match the operation request sent by the RPC requester to the Operation request sent by the RPC executor.

```
POST mg/op/<operation>
```

The <operation> field identifies the module name and rpc identifier string for the desired operation.

For example, if "module-A" defined a "reset" operation, then invoking the operation from "module-A" would be requested as follows:

```
POST example.com/mg/op/module-A:reset
```

If the "rpc" statement has input parameters, then a message-body MAY be sent by the client in the request, otherwise the request message MUST NOT include a message-body.

If the operation is successfully invoked the server MUST send a 2.04 Changed status code. If the operation is not successfully invoked, then a message-body SHOULD be sent by the server, containing an error, as defined in Section 4.7.

If the "rpc" statement has return parameters, then the server invokes a POST method with the same Token value as used in the request from the client. The payload contains the values of the return parameters.

4.4.1. Encoding RPC input parameters

If the "rpc" statement has an "input" section, then the "input" node is provided in the message-body, corresponding to the YANG data definition statements within Request payload.

The following YANG definition is used for the examples in this section.

```
        module example-ops {
         namespace "https://example.com/ns/example-ops";
         prefix "ops";

          rpc reboot {
            input {
              leaf delay {
                units seconds;
                type uint32;
                default 0;
              }
              leaf message { type string; }
              leaf language { type string; }
            }
          }

          rpc get-reboot-info {
            output {
              leaf reboot-time {
                units seconds;
                type uint32;
              }
              leaf message { type string; }
              leaf language { type string; }
            }
          }
        }
```

The client might send the following POST request message:

```
         POST example.com/restconf/operations/example-ops:reboot
         Token:0x56
         Content-Type:
          { "example-ops:input":
            {
              "delay": 600,
              "message": "Going down for system maintenance"
              "language": "en-US"
            }
          }
```

The server may respond with a 2.04 Changed.

4.4.2. Encoding RPC output parameters

If the "rpc" statement has output parameters, then the "output" node is provided in a PUT message, corresponding to the YANG data definition statements within the "output" section.

The "example-ops" YANG module defined Section 4.4 is used for the examples in this section.

The client might send the following POST request message:

```
         POST example.com/mg/op/example-ops:get-reboot-info
   Token: 0x56
```

The server might respond with a POST request that is related to the original RPC invocation by the Token value:

```
         POST [ip:port]/mg/op/example-ops:output
   Token: 0x56
         {"example-ops:output" :
           {
             "reboot-time" : 30,
             "message" : "Going down for system maintenance",
             "language" : "en-US"
           }
         }
```

4.5. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of text need to be transported the datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [I-D.ietf-core-block] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying fragment sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

The block size is specified as exponents of the power 2. The SZX exponent value can have 7 values ranging from 0 to 6 with associated block sizes given by 2**(SZX+4); for example SZX=0 specifies block size 16, and SZX=3 specifies block size 128.

The block number of the block to transmit can be specified. There are two block options: Block1 option for the request payload transported with PUT, POST or PATCH, and the block2 option for the response payload with GET. Block1 and block2 can be combined. Examples showing the use of block option in conjunction with observer options are provided in [I-D.ietf-core-block].

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.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. In the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the variables, the transported results do not correspond with a state having occurred in the server; or worse the returned values are inconsistent. For example: array length does not correspond with actual number of items. It may be advisable to use CBOR maps or CBOR arrays of undefined length which are foreseen for data streaming purposes.

4.6. Resource 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 encoded values of the object identifiers to /.well-known/core with rt="core.mg.data". The available objects identified by the encoded 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 encoded 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"
```

Lists of encoded 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 by reading the YANG module information 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.

The module list can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

```
     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 [I-D.bierman-core-yang-hash]. When a collision occurred in the name space of the target server, a rehash is executed.

4.7. 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.

RESTCONF Status Line	CoAP Status Code
100 Continue	none?
200 OK	2.05
201 Created	2.01
202 Accepted	none?
204 No Content	2.04 Changed
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. 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].

6. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized 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 [RFC6347] for protected access to resources, as well suitable authentication and authorization 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 authorization 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.

7. IANA Considerations

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

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

’rt="core.mg.num-type"’ needs registration with IANA.

’rt="core.mg.srv-type"’ needs registration with IANA.

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

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

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

Content types to be registered:

- application/comi+cbor

8. Acknowledgements

We are very grateful to Bert Greevenbosch who was one of the original authors of the CoMI specification and specified CBOR encoding and use of hashes. 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 Somaraju Abhinav, Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Alexander Pelov, Juergen Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michel Veillette, Michael Verschoor, and Thomas Watteyne.

9. Changelog

Changes from version 00 to version 01

- Focus on MIB only
- Introduced CBOR, JSON, removed BER
- defined mappings from SMI to xx
- Introduced the concept of addressable table rows

Changes from version 01 to version 02

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

- Section on discovery updated

- Text on rehashing introduced

- Elaborated SMI MIB example

- Yang library use described

- use of BigEndian/LittleEndian in Hash generation specified

Changes from version 05 to version 06

- Hash values in payload as hexadecimal and in URL in base64 numbers

- Streamlined CoMI architecture text

- Added select query parameter text

- Data editing optional

- Text on Notify added

- Text on rehashing improved with example

Changes from version 06 to version 07

- reduced payload size by removing JSON hierarchy

- changed rehash handling to support small clients

- added LWM2M comparison

- Notification handling as specified in YANG

- Added Patch function

- Rehashing completely reviewed

- Discover type of YANG name encoding

- Added new resource types

- Read-only servers introduced

- Multiple updates explained

Changes from version 07 to version 08

- Changed YANG Hash algorithm to use module name instead of prefix

- Added rehash bit to allow return values to identify rehashed nodes in the response

- Removed /mg/mod.set resource since this is not needed

- Clarified that YANG Hash is done even for unimplemented objects

- YANG lists transported as CBOR maps of maps

- Adapted examples with more CBOR explanation

- Added CBOR code examples in new appendix

- Possibility to use other than default stream

- Added text and examples for Patch payload

- Repaired some examples

- Added appendices on hash clash probability and hash clash storage overhead

Changes from version 08 to version 09

- Removed hash and YANG to CBOR sections

- removed hashes from examples.

- Added RPC

- Added content query parameter.

- Added default handling.

- Listed differences with RESTCONF

10. References

10.1. Normative References

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

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <http://www.rfc-editor.org/info/rfc6020>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <http://www.rfc-editor.org/info/rfc7396>.

[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, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[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-08 (work in progress), February 2016.

[I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-09 (work in progress), December 2015.

[I-D.vanderstok-core-patch]
Stok, P. and A. Sehgal, "Patch Method for Constrained Application Protocol (CoAP)", draft-vanderstok-core-patch-02 (work in progress), October 2015.

[I-D.ietf-netconf-yang-patch]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-07 (work in progress), December 2015.

[I-D.bierman-core-yang-hash]
Bierman, A. and P. Stok, "YANG Hash", draft-bierman-core-yang-hash-00 (work in progress), February 2016.

10.2. Informative References

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <http://www.rfc-editor.org/info/rfc2578>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <http://www.rfc-editor.org/info/rfc3410>.

[RFC3411] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", STD 62, RFC 3411, DOI 10.17487/RFC3411, December 2002, <http://www.rfc-editor.org/info/rfc3411>.

[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, DOI 10.17487/RFC3414, December 2002, <http://www.rfc-editor.org/info/rfc3414>.

[RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <http://www.rfc-editor.org/info/rfc3416>.

[RFC3418] Presuhn, R., Ed., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3418, DOI 10.17487/RFC3418, December 2002, <http://www.rfc-editor.org/info/rfc3418>.

[RFC4293] Routhier, S., Ed., "Management Information Base for the Internet Protocol (IP)", RFC 4293, DOI 10.17487/RFC4293, April 2006, <http://www.rfc-editor.org/info/rfc4293>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <http://www.rfc-editor.org/info/rfc6643>.

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

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

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

[I-D.ietf-core-interfaces]
Shelby, Z., Vial, M., and M. Koster, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-04 (work in progress), October 2015.

[I-D.ietf-lwig-coap]
Kovatsch, M., Bergmann, O., and C. Bormann, "CoAP Implementation Guidance", draft-ietf-lwig-coap-03 (work in progress), July 2015.

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

[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.

[mibreg] "Structure of Management Information (SMI) Numbers (MIB Module Registrations)", Web http://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml/.

[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.

[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 .

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. Comparison with LWM2M

CoMI and LWM2M, both, provide RESTful device management services over CoAP. Differences between the designs are highlighted in this section.

LWM2M [OMA] objects are defined by standardized numbers. When new types are needed, new numbers need to be defined. This is the major difference with CoMI and YANG, where new modules can incorporate any type that is required without going through a standardization process, but may lead to rehashing. On the one hand LWM2M is static with very small numbered objects, where CoMI with YANG is more dynamic, with a number conversion overhead.

Unlike CoMI, which enables the use of SMIv2 and YANG data models for device management, LWM2M defines a new object resource model. This means that data models need to be redefined in order to use LWM2M. In contrast, CoMI provides access to a large variety of SMIv2 and YANG data modules that can be used immediately.

Objects and resources within CoMI are identified with a YANG hash value, however, each object is described as a link in the CoRE Link Format by LWM2M. This approach by LWM2M can lead to larger complex URIs and more importantly payloads can grow large in size. Using a hash value to represent the objects and resources allows URIs and payloads to be smaller in size, which is important for constrained devices that may not have enough resources to process large messages.

LWM2M encodes payload data in Type-length-value (TLV), JSON or plain text formats. While the TLV encoding is binary and can result in reduced message sizes, JSON and plain text are likely to result in large message sizes when lots of resources are being monitored or configured. Furthermore, CoMI’s use of CBOR gives it an advantage over the LWM2M’s TLV encoding as well since this too is more efficient [citation needed].

CoMI is aligned with RESTCONF for constrained devices and uses YANG data models that have objects containing resources organized in a tree-like structure. On the other hand, LWM2M uses a very flat data model that follows the "object/instance/resource" format, with no possibility to have sub-resources. Complex data models are, as such, harder to model with LWM2M.

In situations where resources need to be modified, CoMI uses the CoAP PATCH operation when resources are modified partially. However, LWM2M uses the CoAP PUT and POST operations, even when a subset of the resource needs modifications.

Authors’ Addresses

Peter van der Stok
consultant

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

Andy Bierman
YumaWorks
685 Cochran St.
Suite #160
Simi Valley, CA 93065
USA

Email: andy@yumaworks.com