core
Internet-Draft
Intended status: Standards Track
Expires: August 13, 2016

A. Bierman
YumaWorks
P. van der Stok
consultant
February 10, 2016

YANG Hash
draft-bierman-core-yang-hash-00

Abstract

This document describes an encoding of YANG names to 30 bit hashes. This document extends the CoMI draft to reduce payload and URI of CoMI network requests. The technique can be applied to other YANG based applications to reduce the payload by replacing the YANG names with 30 bit numbers.

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.

The drafts [I-D.ietf-netconf-restconf] and [I-D.vanderstok-core-comi] describe REST-like interfaces to access structured data defined in YANG [RFC6020].

The payload format CBOR [RFC7049] can be used to reduce the size of the transported payload. In that case the size of the payload depends for a large part on the YANG names. Reducing the names significantly reduces the payload size further (see Appendix C). This draft proposes a hashing technique to encode the YANG names into 30 bit numbers.

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 [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 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.

Rehash bit: Bit 31. If a particular YANG hash value is a re-hash for an identifier, then the rehash bit will be set in the object identifier. This allows the server to return descendant nodes that have been rehashed, instead of returning an error for an entire GET request.

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.

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. 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 [murmur3]. Implementations are available online [murmur-imp]. When converting 4 input bytes to a 32-bit integer in the hash algorithm, the Little-Endian convention MUST be used.

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 Section 3 is executed.

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

The module name is used to identify the namespace of the object node. The prefix cannot be used because it is allowed to change over time. The module name is never allowed to change.

The module name MUST be present in the identifier for the first node in the object path identifier.

If a child node in the object path identifier is from the same module namespace as its parent, then the module-name MUST NOT be used in the identifier.

If a child node in the object path identifier is not from the same module namespace as its parent, then the module-name MUST be used in the identifier.

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 YANG Hash value is calculated for all data nodes in the module, even if the server only implements a subset of these objects. This includes all "data-def", "rpc", "notification", and external data nodes derived from "augment" statements.

Example: the following canonical identifier is used for the 'mtu' leaf in the ietf-interfaces module:

```
/ietf-interfaces:interfaces/interface/mtu
```

Example: the following canonical identifier is used for the 'ipv4' container in the ietf-ip module, which augments the 'interface' list in the ietf-interfaces module:

```
/ietf-interfaces:interfaces/interface/ietf-ip:ipv4
```

3. Re-Hash Error Procedure

In most cases, the hash function is expected to produce unique values for all the node names supported by a constrained server. 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 need to be considered (see Appendix A for clash probabilities). 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 occur at a given server dependent on the set of modules supported by the server. The client needs to discover any re-hash mappings on a per server basis.

If the server needs to re-hash any YANG name, then it MUST create a "rehash" entry for all its rehashed node names, as described in Section 4.

A re-hashed object identifier has the rehash bit set in the identifier, every time it is sent from the server to the client. This allows the client to identify nodes for which a "reverse rehash" entry may need to be retrieved (see Section 6). A client does not need to retrieve the rehash map before retrieving or configuring rehashed data nodes.

If any node identifier provided by the client is not available because it has been rehashed, the server MUST return a rehash error, containing the 'rehash' entries for all the invalid nodes which were specified by the client.

It is possible that none of the node identifiers provided by the client in a GET method are invalid and rehashed, but rather one or more descendant nodes within the selected subtree(s) have been rehashed. In this case, a rehash error is not returned. Instead the requested subtree(s) are returned, and the rehash bit is set for any descendant node(s) that have been rehashed. The client will strip off the rehash bit and retrieve the 'revhash' entry for these nodes (if not already done).

4. Re-Hashing Names Procedure

A hash collision occurs if two different path identifier strings have the same hash value. If the server has around 1000 node names in its YANG modules, then the probability of a collision is a half per mil (see Appendix A). If a hash collision occurs on the server, then the node name 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.

For example, rehashing could be done by prefixing a "˜" character in front of the clashing name and execute murmur3 on the thus modified name. If necessary the prefixing can be done multiple times until the clashes are resolved. Using the prefixing is not needed from an inter-operability point of view but provides a procedure for client and server to calculate the rehash without reading the "rehash" entry.

5. 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 object*
               +--ro module     string
               +--ro newhash    uint32
               +--ro path?      string
```

```
   <CODE BEGINS> file "ietf-yang-hash@2016-02-10.yang"

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

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

        Copyright (c) 2016 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-bierman-core-yang-hash-00.txt

     // RFC Ed.: update the date below with the date of RFC publication
     // and remove this note.
     revision 2016-02-10 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: YANG Hash.";
     }

     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.  This hash value
              cannot be used on the server.  The rehashed
              value for each affected object must be used instead.";
         }

         list object {
           min-elements 2;

           description
             "Each entry identifies one of the objects involved in the
              hash collision and contains the rehash information for
              that object.";

           leaf module {
             type string;
             mandatory true;
             description
               "The module name identifying the module namespace
                for this object.";
           }

           leaf newhash {
             type uint32;
             mandatory true;
             description
               "The new hash value for this object. The rehash bit is
                not set in this value.";
           }

           leaf path {
             type string;
             description
               "The object path identifier string used in the original
                YANG hash calculation. This object MUST be included for
                any objects in the rehash entry with the same 'module'
                value.";
           }
         }
       }
     }

   }

   <CODE ENDS>
```

6. YANG Re-Hash Examples

In this example there are two YANG modules, "foo" and "bar".

```
   module foo {
     namespace "http://example.com/ns/foo";
     prefix "f";
```

```
       revision 2015-06-07;

       container A {
         list B {
           key name;
           leaf name { type string; }
           leaf counter1 { type uint32; }
         }
       }
     }

     module bar {
       namespace "http://example.com/ns/bar";
       prefix "b";
       import foo { prefix f; }
       revision 2015-06-07;

       augment /f:A/f:B {
         leaf counter2 { type uint32; }
       }
     }
```

This set of 3 YANG modules containing a total of 7 objects produces the following object list. Note that actual hash values are not shown, since these modules do not actually cause the YANG Hash clashes described in the examples.

```
        Object    Path                     Hash

     foo:

        container /foo:A                   h1
        list      /foo:A/B                 h2
        leaf      /foo:A/B/name            h3
        leaf      /foo:A/B/counter1        h4

     bar:

       leaf      /foo:A/B/bar1:counter2   h5
```

6.1. Multiple Modules

In this example, assume that the ’B’ and ’counter2’ objects produce the same hash value, so ’h2’ and ’h5’ both have the same value (e.g. ’1234’):

The client might retrieve an entry from the list "/foo:A/B", which would cause this subtree to be returned. Instead, the server will return a message with the resource type "core.mg.yang-hash", representing the "yang-hash" data structure. Only the entry for the requested identifier is returned, even if multiple ’rehash’ list entries exist.

```
REQ: GET example.com/mg/h2?keys="entry2"

RES: 4.00 "Bad Request" (Content-Format: application/cbor)
{
   "ietf-yang-hash:yang-hash" : {
     "rehash" : [
        {
          "hash" : 1234,
           "object" : [
             {
               "module" : "foo",
               "newhash" : 5678
             },
             {
               "module" : "bar",
               "newhash" : 8182
             }
           ]
        }
     ]
  }
}
```

6.2. Same Module

In this example, assume that the ’B’, ’counter1’, and ’counter2’ objects produce the same hash value, so ’h2’, ’h4’, and ’h5’ objects all have the same value (e.g. ’1234’):

The client might retrieve an entry from the list "/foo:A/B", which would cause this subtree to be returned. Instead, the server will return a message with the resource type "core.mg.yang-hash", representing the "yang-hash" data structure. Only the entry for the requested identifier is returned, even if multiple ’rehash’ list entries exist.

```
REQ: GET example.com/mg/h2?keys="entry2"

RES: 4.00 "Bad Request" (Content-Format: application/cbor)
{
   "ietf-yang-hash:yang-hash" : {
     "rehash" : [
        {
          "hash" : 1234,
           "object" : [
             {
               "module" : "foo",
               "newhash" : 5678,
               "path" : "/foo:A/B"

             },
             {
               "module" : "foo",
               "newhash" : 2134,
               "path" : "/foo:A/B/counter1"
             },
             {
               "module" : "bar",
               "newhash" : 8182,
               "path" : "/foo:A/B/bar:counter2"
             }
           ]
        }
     ]
   }
}
```

7. Retrieval of Rehashed Data

In this example, assume that the ’B’, ’counter1’, and ’counter2’ objects produce the same hash value, so ’h2’, ’h4’, and ’h5’ objects all have the same value (e.g. ’1234’):

The client might retrieve the top-level container "/foo:A", which would cause this subtree to be returned. Since the identifier (h1) has not been re-hashed, the server will return the requested data. The new hashes for ’h2’, ’h4’,and ’h5’ will be returned, except the rehash bit will be set for these identifiers.

The notation "R+" indicates that the rehash bit is set.

```
REQ: GET example.com/mg/h1

RES: 2.05 Content (Content-Format: application/cbor)
{
     h1 : {
        R+5678 : {
           { h3 : "entry1"}:
             {R+2134: 615,
              R+8182: 7},
           { h3 : "entry2"}:
             {R+2134: 491,
              R+8182: 26}
         }
      }
}
```

The client will notice that the rehash bit is set for 3 nodes. The client will need to retrieve the full "yang-hash" container at this point, if that has not already been done. The rehashed identifiers will be in "rehash" list, contained in the "newhash" leaf for the "object" list.

8. YANG Hash representations

YANG hashes are represented in two fashions.

8.1. YANG Hash in payload

When a YANG hash value is printed in the payload, error-path or other string, then the lowercase hexadecimal representation is used. Leading zeros are used so the value uses 8 hex characters.

8.2. 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".

9. YANG Hash Examples

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

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

The 30 bit murmur3 hash value (see Section 2) is calculated on this string with hash: 0x047c468b and EfEaM. The request using this hash value is shown below:

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

RES: 2.05 Content (Content-Format: application/cbor)
{
    0x047c468b : "2014-10-26T12:16:31Z"
}
```

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):

```
/ietf-system:system-state/clock (0x021ca491 and CDKSQ)
/ietf-system:system-state/clock/current-datetime (0x047c468b)
/ietf-system:system-state/clock/boot-datetime (0x1fb5f4f8)
```

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):

```
/ietf-interfaces:interfaces/interface/ietf-ip:ipv6/neighbor
    (0x2445e478 and kReR4)
/ietf-interfaces:interfaces/interface/ietf-ip:ipv6/neighbor/ip
    (0x2283ed40 and ig-la)
/ietf-interfaces:interfaces/interface/ietf-ip:ipv6/neighbor/
    link-layer-address  (0x3d6915c7)
```

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 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/ipNetToPhysicalTable (0x0aba15cc and kuhXM)
   /IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry
      (0xo6aaddbc and Gqt28)
   /IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
      ipNetToPhysicalIfIndex (0x346b3071)
   /IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
      ipNetToPhysicalNetAddressType (0x3650bb64)
   /IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
      ipNetToPhysicalNetAddress (0x06fd4d91)
   /IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
      ipNetToPhysicalPhysAddress (0x26180bcb)
   /IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
      ipNetToPhysicalLastUpdated (0x3d6bbe90)
   /IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
      ipNetToPhysicalType (0x35ecbb3d)
   /IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
      ipNetToPhysicalState (0x13038bb5)
   /IP-MIB:IP-MIB/ipNetToPhysicalTable/ipNetToPhysicalEntry/
      ipNetToPhysicalRowStatus (0x09e1fa37)
```

The YANG Hash values for the YANG Patch request objects are calculated as follows:

```
   0x2c3f93c7: /ietf-yang-patch:yang-patch
   0x2fb8873e: /ietf-yang-patch:yang-patch/patch-id
   0x011640f0: /ietf-yang-patch:yang-patch/comment
   0x16804b72: /ietf-yang-patch:yang-patch/edit
   0x2bd93228: /ietf-yang-patch:yang-patch/edit/edit-id
   0x1959d8c9: /ietf-yang-patch:yang-patch/edit/operation
   0x1346e0aa: /ietf-yang-patch:yang-patch/edit/target
   0x0750e196: /ietf-yang-patch:yang-patch/edit/point
   0x0b45277e: /ietf-yang-patch:yang-patch/edit/where
   0x2822c407: /ietf-yang-patch:yang-patch/edit/value
```

10. Security Considerations

The replacement of name-strings by numbers does not affect the security of the transmitted requests.

11. IANA Considerations

No considerations for IANA apply.

12. Acknowledgements

We are very grateful to Bert Greevenbosch who suggested the use of hashes and specified the use of murmur3. Many thanks for their contributions go to Alexander Pelov, Juergen Schonwalder, Anuj Sehgal and Michel Veillette.

This material is based upon work supported by the The Space & Terrestrial Communications Directorate (S&TCD) under Contract No. W15P7T-13-C-A616. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of The Space & Terrestrial Communications Directorate (S&TCD) .

13. Changelog

Version 0 is extracted from comi draft version 8 [I-D.vanderstok-core-comi]. Changed Appendix A, and added Appendix C.

14. References

14.1. 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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[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>.

[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>.

[murmur3] , "murmurhash family", Web http://en.wikipedia.org/wiki/MurmurHash, .

[murmur-imp] , "murmurhash implementation", Web https://code.google.com/p/smhasher/, .

14.2. 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>.

[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>.

[I-D.ietf-netconf-restconf] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-07 (work in progress), July 2015.

[I-D.vanderstok-core-comi] Stok, P., Bierman, A., Schoenwaelder, J., and A. Sehgal, "CoAP Management Interface", draft-vanderstok-core-comi-08 (work in progress), October 2015.

[coll-prob] Preshing, j., "Hash collision probabilities", Web http://preshing.com/20110504/hash-collision-probabilities, May 2011.

[birthday] Wikipedia, , "Birthday problem", Web https://en.wikipedia.org/wiki/Birthday_problem, .

Appendix A. Hash clash probability

Number of names	28 bits	29 bits	30 bits	31 bits	32 bits	33 bits
10	1,7E-07	8,4E-08	4,2E-08	2,1E-08	1,1E-08	5,2E-09
100	1,8E-05	9,2E-06	4,6E-06	2,3E-06	1,2E-06	5,8E-07

200	7,4E-05	3,7E-05	1,9E-05	9,3E-06	4,6E-06	2,3E-06
10^3	1,9E-03	9,3E-04	4,7E-04	2,3E-04	1,2E-04	5,8E-05
4000	3,0E-02	1,5E-02	7,5E-03	3,7E-03	1,9E-03	9,3E-04
10^4	1,9E-01	9,3E-02	4,6E-02	2,3E-02	1,2E-02	5,8E-03

Table 1: Probability of one or more clashes

This appendix calculates the probability of a hash clash as function of the hash size and the number of YANG names. The standard way to calculate the probability of a clash is to calculate the probability that no clashes occur [birthday], [coll-prob].

The probability of no clashes when generating k numbers with a hash size of N=2^bits is given by:

D(N,k) = ((N-1)/N)*((N-2)/N)*....(N-(k-1))/N (1)

which can be approximated with:

exp(-k*(k-1)/2N) (2)

The probability that one or more clashes occur is given by:

1 - exp(-k*(k-1)/2N) ~ k*(k-1)/2N (3)

Table 1 shows the probabilities for a given set of values of N=2^bits and number of YANG node names k. Probabilities which are larger than 0.5 are not shown because the used approximations are not accurate any more.

The overhead in servers and clients depends on the number of clashes. Therefore it is interesting to know the probability that more than one clash occurs. The probability of generating k numbers with a hash size of N=2^bits, where 2 numbers are identical and all the rest is different, is composed of the following parts. The probability that the second number is equal to the first is 1/N. The possible number of configurations of 2 equal numbers out of k is given by SUM_i=1,k-1 (i). The probability of k-1 different numbers is given by D(N,k-1). The probability of generating exactly one clash of two numbers is given by:

(SUM_1,k-1 (i))*D(N,k-1)/N

Where we used formula (1). Working out the summation and using (2), the probability that exactly one pair of hashes clashes is given by:

(k*(k-1)/2N)*exp(-(k-1)*(k-2)/2N)

The probability that more than one pair clashes is given by the probability that a clash occurs minus the probability that only one pair clashes. This leads to:

1 - exp(-k*(k-1)/2N) - (k*(k-1)/2N)*exp(-(k-1)*(k-2)/2N)

Substituting formula 3, gives:

k*(k-1)/2N - k*(k-1)/2N + (k*(k-1)^2*(k-2)/4N^2 =

(k*(k-1)^2*(k-2)/4N^2

Number of names	28 bits	29 bits	30 bits	31 bits	32 bits	33 bits
10	2,3E-14	5,6E-15	1,4E-15	3,5E-16	8,8E-17	2,2E-17
100	3,3E-10	8,3E-11	2,1E-11	5,2E-12	1,3E-12	3,3E-13
200	5,4E-09	1,4E-09	3,4E-10	8,5E-1	2,1E-11	5,3E-12
10^3	3,5E-06	8,6E-07	2,2E-07	5,4E-08	1,4E-08	3,4E-09
4000	8,9E-04	2,2E-04	5,6E-05	1,4E-05	3,5E-06	8,7E-0

						7
10^4	3,5E-02	8,7E-03	2,2E-03	5,4E-04	1,4E-04	3,4E-0 5

Table 2: Probability of more than 2 entries equal clashes

The corresponding probabilities are shown in Table 2. Assuming a hash size of 2^30, and about 1000 YANG nodes in a server, the probability of one clashing pair is 0.5*10^-3, and the probability that more clashes occur is 2*10^-7.

Appendix B. Hash clash storage overhead

Clashes may occur in servers dynamically during the operation of their clients, and clashes must be handled on a per server basis in the client. When rehashing is possible, clashing names on a given server are prefixed with a character (for example "~") and are rehashed, thus leading to hash values which uniquely identify the data nodes in the server. This appendix calculates the storage space needed when a clash occurs in a set of servers running the same server code. Appendix A shows that more than one clash in a server set is exceptional, which suggests at most two clashing object names in a given server.

The sizes of server and client tables needed to handle the clashes in client and server are calculated separately, because they differ significantly.

B.1. Server tables

When a request arrives at the server, the server must relate the incoming hash value to the memory locations where the related values are stored. In the server a translation table must be provided that relates a hash value to a memory address where either the raw data or a description of the data (as prescribed by the YANG compiler) are stored. The required storage space is a sequence of (32 bit yang hash, 64 bit memory address) for every YANG data node. The translation table size in a server is 12 bytes times the number of YANG data nodes in the server.

For every clashing hash value the following server clash table entries are needed: Clashed hash value, module name, and new hash. To reduce table size in the client, module name can be replaced with a 1 byte module identifier. The module identifier represents the index value of an array of module names. Server clash table size is: 2 hashes (8 bytes) + 1 module identifier (1 byte)

B.2. Client tables

In the client, the compiled code must refer to a hash value. To cope with on-the-fly rehashing, the compiled code needs to invoke a procedure that returns the possibly rehashed value as function of the original hash value, module name, and server address. The client needs to store a client clash table containing: the clashed hash value, module name, server IPv6 address (or name), and rehash value for as many rehashes occurring in a given server. Many servers contain an identical set of YANG modules. The servers containing the same module set belong to the same server type. The server type is used to administrate the hash clash occurrence. To reduce client clash table size, module name can be replaced with a 1 byte module identifier. The module identifier represents the index value of an array of module names. A table of IPv6 server addresses must already exist in the client. To reduce client clash table size further, the server IPv6 address can be replaced with a 1 byte server type identifier. The server table can be ordered according to server type. A table with server type and pointer to sub-table start suffices to find all IPv6 addresses belonging to a server type.

The client clash table reduces to clashed hash value (4 bytes), module identifier (1 byte), server type identifier (1 byte) and rehash value (4 bytes).

B.3. Table summary

Sizes of all the tables are:

Server clash table: 9 bytes per clashing object name.

Client clash table: 10 bytes per server type, per clashing object name.

Array of module names: Sum of module name sizes.

Server identifier table: 1 byte server type + 4 bytes pointer per server type.

The existence of the translation table in a server is required independent of rehashing. The table sizes calculated to estimate the storage requirements coming from CoMI clashes. Assume the following numbers:

- 500 data nodes per server

- 10 server types

- 30 modules
- Module name is on average 20 bytes
- Maximum of 2 clashing object names occurring in 2 server types

This yields the following overhead estimates:

Server tables size:

  * Server clash table: 2*9 bytes represents 18 bytes
  * Module name array: 30*20 represents 600 bytes

Client table sizes:

  * Client clash table: 10*2*2 represents 40 bytes for 2 object names in 2 server types.
  * Module name array: 30*20 represents 600 bytes.
  * Server identifier table: 10*5 = 50 bytes

In conclusion:

1. Storage space size in client is independent of number of servers but depends on number of server types.
2. There is a common storage size for the module array of 600 bytes.
3. Assuming 2 clashing object names in 2 server types, additional storage space in client is 40 bytes and in server 18 bytes.
4. When the module array is suppressed (removing 600 bytes storage space), the server clash table and the client clash table increase with 40 bytes and 80 bytes respectively.

Appendix C. payload reduction

The hashing of the YANG identifier in the transported payload reduces the size of the YANG objects transported in the payload. This note calculates the payload size reduction and the number of YANG objects that can be transported in a single 802.154 CoAP packet. The payload is assumed to be a sequence of maps, where the entry ("ident" : value) is referred to as a map, as shown below. The value can be an integer or a string.

```
{
     "ident1" : value 1,
     "ident2" : value 2.
     Etc .....
}
```

In general, we can assume that the payload size (SI) of a YANG identifier string lies between 6 to 128 bytes with an average of 30-40 bytes. The payload size (SV) of a value can range between 1 byte to 128 bytes. The transport format is CBOR. The overhead coming from CBOR is composed of:

- o One byte to indicate the number of maps (> 2 maps in the example above).

- o Two bytes per map: one describing the identifier, and one describing the value.

When the CBOR byte describes an integer (major type 0), the size of the following integer is 0, when integer < 24. Otherwise the size of the following integer is 1 to 8 bytes. The size of the string remains unchanged when preceded by a CBOR byte (major type 2). When the string size < 24 , no extra bytes are needed; when the string size lies between 24 and 128 one extra CBOR byte overhead is needed. The parameters SE, SV and SI are introduced with the following meaning:

- o SE is the size of the identifier encoded by a hash or other unsigned integer, with 0 < SE < 5; because the hash size is 4 bytes and the minimum managed encoded identifier is assumed to take 1 byte.

- o SV is the size of the value, with 0 <= SV < 128.

- o SI is the size of the original YANG hash identifier string, with 4 < SI < 64.

The impact of the conversion from YANG identifier string to unsigned integer is straightforward to calculate per map. It is assumed that one CBOR byte or two CBOR bytes are needed dependent on the sizes SI and SV. A map size with the original YANG identifier string is given by:

- o Map size is: 2 + SI + SV, when SI, SV < 24;

- o Map size is: 3 + SI + SV, when SI < 24 and SV > 23, or SI > 23 and SV < 24;

- Map size is: 4 + SI + SV, when SI, SV > 23.

The map size when SI is converted to an unsigned integer with size SE is given by:

- Encoded map size is: 2 + SE + SV, when SV < 24;
- Encoded map size is: 3 + SE + SV, when SV > 23.

The improvement of the conversion can be written as

- (2+SE+SV)/(2+SI+SV) when SI, SV < 24;
- (2+SE+SV)(3+SI+SV) when SI > 23 and SV < 24;
- (3+SE+SV)/(3+SI+SV) when SI < 24, and SV > 23;
- (3+SE+SV)/(4+SI+SV) when SI, SV > 23.

Table 3 shows the size reduction for different SI and SV values and SE = 4:

SV->	0	4	10	20	30	40	60
SI=6	0,75	0,83	0,89	0,93	0,95	0,96	0,97
SI=10	0,83	0,88	0,91	0,94	0,95	0,96	0,97
SI=20	0,91	0,92	0,94	0,95	0,96	0,97	0,98
SI=30	0,97	0,97	0,98	0,98	0,98	0,99	0,99
SI=40	0,98	0,98	0,98	0,98	0,99	0,99	0,99

Table 3: Payload size reduction as function of SI and SV

Another way to look at it is to see how many maps fit in a packet. Assuming a 802.15.4 packet with short addresses, the IEEE header is 13 bytes. The CoAP header assuming only mesh traffic takes 6 bytes. The URI is composed of 3 options, where each option header takes 1 byte: total of 3 bytes. The URI is assumed to be split up in the following 3 parts:

- URI-host "example.net" takes 13 bytes, which necessitates 1 length byte: total of 14 bytes.

- URI-path = "mg" takes 4 bytes.

- URI-path = "hash value" takes 7 bytes.

Total URI takes 3+14+4+7 = 28 bytes. For the CoMI payload, there remains 127 -13 - 6 - 28 = 80 bytes. Subtracting the byte indicating the number of CBOR maps, the payload size for the maps is 79 bytes.

N.B. no security overhead is included.

When the YANG string identifier needs to be stored, the number of storable maps is given by:

- 79/(2+SI+SV) for SI, SV < 24;

- 79/(3+SI+SV) for SI < 24 and SV > 23, or SI > 23 and SV < 24;

- 79/(4+SI+SV) for SI, SV > 23.

Table 4 shows the number of transported maps for different values of SI and SV.

SV->	0	4	10	20	30	40	60
SI=6	9	6	4	2	2	1	1
SI=10	6	4	3	2	1	1	1
SI=20	3	3	2	1	1	1	0
SI=30	2	2	1	1	1	1	0
SI=40	1	1	1	1	1	0	0

Table 4: Nr of transported maps as function of SI and SV

Assuming the encoding of the YANG identifier to an unsigned integer with size 0 < SE < 5, the number of storable maps is given by:

- 79/(2+SE+SV) for SV < 24;

- 79/(3+SE+SV) for SV > 23.

Table 5 shows the number of transported maps for different values of SE and SV, independent of SI.

SV->	0	4	10	20	30	40	60
SE=1	26	11	6	3	2	1	1
SE=2	19	9	5	3	2	1	1
SE=3	15	8	5	3	2	1	1
SE=4	13	7	4	3	2	1	1

Table 5: Nr of transported maps as function of SE and SV,

The value of SI for the average YANG string identifier is 30. When the size of the value part of the map is less than 10 (SV < 10) and SI=30, the impact of the hashing is significant: 10 maps can be transported instead of 1 or 2. Further reducing the value of SE from 4 to 1 increases the number of transported maps with a factor 2 to 1,2.

Authors' Addresses

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

Email: andy@yumaworks.com

Peter van der Stok
consultant

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