Network Working Group
Internet-Draft
Updates: 7950 (if approved)
Intended status: Standards Track
Expires: September 12, 2019

B. Claise
J. Clarke
R. Rahman
R. Wilton, Ed.
Cisco Systems, Inc.
B. Lengyel
Ericsson
J. Sterne
Nokia
K. D'Souza
AT&T
March 11, 2019

YANG Semantic Versioning for Modules
draft-verdt-netmod-yang-semver-00

Abstract

This document specifies a new YANG module update procedure using semantic version numbers, to allow for limited non-backwards-compatible changes, as an alternative proposal to module update rules in the YANG 1.1 specifications. This document updates RFC 7950, RFC 8407 and RFC 8525.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a solution to the YANG module lifecycle problems described in [I-D.verdt-netmod-yang-versioning-reqs], covering all of the specified requirements except for requirements: 2.2, 3.1, and 3.2.

Specifically, this document recognises a need to sometimes allow YANG modules to evolve with non-backwards-compatible changes, which might end up breaking clients. The solution makes use of semantic version numbers to help manage the lifecycle of YANG modules.

The solution is comprised of the following seven parts:

A definition for the YANG semantic versioning scheme for modules, and an explanation of how the semver extension can be used to annotate modules with their semantic version number.

A YANG extension to allow YANG module imports to be restricted to modules with particular semantic versions, allowing inter-module version dependencies to be captured within YANG module definitions.

Updates to the YANG 1.1 module update rules to accommodate the semantic versioning scheme.

Updates and augmentations to ietf-yang-library to include the YANG semantic version number in the module descriptions, to report how 'deprecated' and 'obsolete' nodes are handled by a server, and to clarify how module imports are resolved when multiple versions could otherwise be chosen.

A YANG extension to add a 'description' statement to the YANG 'status' statement to allow additional documentation as to why a node is being deprecated, and what alternatives may be available.

A description of how YANG semantic versioning applies to YANG instance data.

Guidelines to YANG module authors on how the YANG semantic versioning rules should be used, along with examples.

Open issues are listed at Appendix A.1, and tracked at <https://github.com/netmod-wg/yang-ver-dt/issues>.

1.1. Updates to YANG RFCs

1.1.1. Updates to RFC7950

This document proposes updates to [RFC7950] to address some of the requirements. It should be noted that there is also active WG discussion on the next steps towards an updated version of YANG, and potentially some of the functionality described here could be folded into an updated revision of [RFC7950], although that might adversely impact when (parts of) a standards based YANG module versioning solution is available.

The sections listed below provide updates to [RFC7950]. The design team does not believe any of the changes require a new version of the YANG language. It is believed that the extensions as they are defined can coexist with existing YANG 1.1 clients.

o Section 4 describes modification to the [RFC7950] Section 11 module update text to advise the use of semantic versioning as described in this document.

o Section 3 describes an extension to do import by semantic version.

o Section 6 defines an extension that adds a description child element to the YANG "status" statement.

1.1.2. Updates to RFC8525

This document updates [RFC8525]. Section 5 defines how a reader of a YANG library datastore schema chooses which version of an import-only module is used to resolve a module import when the definition is otherwise ambiguous.

1.1.3. Updates to RFC8407

Section 8 updates [RFC8407] to provide guidelines on how the YANG module semantic versioning can be used to manage the lifecycle of YANG modules when using strict RFC 7950 chapter 11 backwards compatibility rules are not pragmatic.

1.2. Complementary solutions for the other requirements

This section is to aid the WG understand how the full set of YANG versioning requirements are intended to be holistically addressed and is intended to be removed if this draft is adopted by the WG.

As stated previously, this draft does not address requirements 2.2, 3.1 and 3.2 of the requirements specified in [I-D.verdt-netmod-yang-versioning-reqs]. Instead, additional work is needed to address those requirements, which the design team believes would be best addressed in separate drafts. It is hoped that the WG agrees that viable solutions to the other requirements exist that complement the solution proposed in this draft, and thus this work can usefully progress in parallel. In particular, there is value to the industry to achieve standardization of a partial solution that addresses the majority, but not all, of the stated requirements, on the agreement that a full solution will follow.

The two additional drafts are:

A tooling based solution is proposed for requirement 2.2, that allows two YANG schema versions to be algorithmically compared, with the algorithm reporting the list of differences between the two YANG schema and whether each change is regarded as being editorial, backwards-compatible, or non-backwards-compatible. Annotations to the YANG modules, via the use of extension statements, may help improve the accuracy of the comparison algorithm, particularly for statements that are very hard for an algorithm to correctly classify the scope of any differences (e.g., a change in the semantic behaviour of a data node defined via modifications to the associated YANG description statement). Given that requirement 2.2 is a soft requirement (SHOULD rather than MUST), and practical experience with the tooling is required, it is proposed that this work is deferred at this time.

A proposed solution for requirements 3.1 and 3.2 is via the use of YANG packages [I-D.rwilton-netmod-yang-packages] and a protocol based version selection scheme that can be used by clients to choose a particular YANG datastore schema from the set of datastore schema that are supported by the server.

2. YANG Semantic Versioning

The chapter defines YANG Semantic Versioning, explains how it is used with YANG modules, and the rules associated with changing a module's semantic version number when the module definitions are updated.

The YANG semantic versioning scheme applies only to YANG modules. YANG submodules are not independently versioned by the YANG semantic versioning scheme. Instead, if a versioned module includes one or more submodules then those submodules are implicitly versioned as part of the module's 'semver:version' statements, and all the module's 'include' statements MUST specify the revision-date for each of the included submodules.

2.1. Classification of changes between module revisions

The principle aim of YANG semantic versioning is to allow a user of a YANG module to understand the overall significance of any changes between two module revisions solely based on the semantic version number.

The semantic version change between any two arbitrary revisions of a YANG module can be classified into one of four categories: 'unchanged', 'editorial, 'backwards-compatible' or 'non-backwards-compatible'. A summary of the classification is given below, with the specific rules as they apply to YANG statements provided in Section 4.

The semantic version change between two module revisions is defined as 'unchanged' if, after excluding 'revision' and 'semver:version' statements and their substatements, the only remaining changes are insignificant white space changes.

An 'editorial' module semantic version change is where there are changes in the module's statements, between the two module revisions, but those changes do not affect the syntax or semantic meaning of the module in any way. An example of an editorial change would be a fix to a spelling mistake in a description statement.

A 'backwards-compatible' module semantic version change is where some syntax or semantic changes exists between the two module revisions, but all changes follow the rules specified in Section 4.2.

A 'non-backwards-compatible' module semantic version change is where some syntax or semantic changes exists between the two

module revisions, and those changes do not follow the rules for a 'backwards-compatible' version change.

2.2. YANG Semantic Versioning Scheme for Modules

This document defines the YANG semantic versioning scheme that is used for YANG modules. The versioning scheme has the following properties:

The YANG semantic versioning scheme is extended from version 2.0.0 of the semantic versioning scheme defined at semver.org [semver] to cover the additional requirements for the management of YANG module lifecyles that cannot be addressed using the semver.org 2.0.0 versioning scheme alone.

Unlike the semver.org 2.0.0 versioning scheme, the YANG semantic versioning scheme supports limited updates to older versions of YANG modules, to allow for bug fixes and enhancements to module versions that are not the latest.

Module definitions that follow the semver.org 2.0.0 versioning scheme are fully compatible with implementations that understand the YANG semantic versioning scheme.

If module updates are always restricted to the latest version of the module only, then the version numbers used by the YANG semantic versioning scheme are exactly the same as those defined by the semver.org 2.0.0 versioning scheme.

Every YANG module versioned using the YANG semantic versioning scheme specifies the module's semantic version number by including the 'semver:module-version' statement according to the following rules:

The module MUST include at least one revision statement.

The most recent module revision statement MUST include a 'semver:module-version' sub-statement, that defines the module's YANG semantic version.

The preceding module revision statement SHOULD also include a 'semver:module-version' sub-statement, to allow the module's semantic version history to be derived.

All other revision statements MAY include a 'semver:module-version' sub-statement if they have an associated YANG semantic version.

"The YANG semver version number is expressed as a string of the form: 'X.Y.Zv'; where X, Y, and Z each represent non-negative integers smaller than 32768, and v represents an optional single character suffix: 'm' or 'M'.

o 'X' is the MAJOR version. Changes in the major version number indicate changes that are non-backwards-compatible to versions with a lower major version number.

o 'Y' is the MINOR version. Changes in the minor version number indicate changes that are backwards-compatible to versions with the same major version number, but a lower minor version number and no patch 'm' or 'M' modifier.

o 'Zv' is the PATCH version and modifier. Changes in the patch version number can indicate editorial, backwards-compatible, or non-backwards-compatible changes relative to versions with the same major and minor version numbers, but lower patch version number, depending on what form modifier 'v' takes:

   * If the modifier letter is absent, the change represents an editorial change

   * 'm' - the change represents a backwards-compatible change

   * 'M' - the change represents a non-backwards-compatible change

The YANG module name and YANG semantic version number uniquely identifies a revision of a module, with an associated revision date. There MUST NOT be multiple instances of a YANG module definition with the same module name and YANG semantic version number but different content or revision date.

There MUST NOT be multiple versions of a YANG module that have the same MAJOR, MINOR and PATCH version numbers, but different patch modifier letter. E.g., module version "1.2.3M" MUST NOT be defined if module version "1.2.3" has already been defined.

2.2.1. Examples for YANG semantic version numbers

The following diagram and explanation illustrates how YANG semantic version numbers work.

Example YANG semantic version numbers for an example module:

```
        0.1.0
          |
        0.2.0
          |
        1.0.0
          |  \
          |   1.1.0 -> 1.1.1m -> 1.1.2M
          |    |
          |   1.2.0 -> 1.2.1M -> 1.2.2M
          |    |
          |   1.3.0 -> 1.3.1
          |
        2.0.0
          |
        3.0.0
             \
              3.1.0
```

The tree diagram above illustrates how an example modules version history might evolve. For example, the tree might represent the following changes, listed in chronological order from oldest revision to newest:

0.1.0 - first beta module version

0.2.0 - second beta module version (with NBC changes)

1.0.0 - first release (may have NBC changes from 0.2.0)

1.1.0 - added new functionality, leaf "foo" (BC)

1.2.0 - added new functionality, leaf "baz" (BC)

1.3.0 - improve existing functionality, added leaf "foo-64" (BC)

1.3.1 - improve description wording for "foo-64" (Editorial)

1.1.1m - backport "foo-64" leaf to 1.1.x to avoid implementing "baz" from 1.2.0 (BC)

2.0.0 - change existing model for performance reasons, e.g. re-key list (NBC)

1.1.2M - NBC point bug fix, not required in 2.0.0 due to model changes (NBC)

3.0.0 - NBC bugfix, rename "baz" to "bar"; also add new BC leaf "wibble"; (NBC)

1.2.1M - backport NBC fix, changing "baz" to "bar"

1.2.2M - backport "wibble". This is a BC change but "M" modifier is sticky.

3.1.0 - introduce new leaf "wobble" (BC)

The partial ordering relationships based on the semantic versioning numbers can be defined as follows:

1.0.0 < 1.1.0 < 1.2.0 < 1.3.0 < 2.0.0 < 3.0.0 < 3.1.0

1.0.0 < 1.1.0 < 1.1.1m < 1.1.2M

1.0.0 < 1.1.0 < 1.2.0 < 1.2.1M < 1.2.2M

There is no ordering relationship between 1.1.1M and either 1.2.0 or 1.2.1M, except that they share the common ancestor of 1.1.0.

Looking at the version number alone, the module definition in 2.0.0 does not necessarily contain the contents of 1.3.0. However, the module revision history in 2.0.0 would likely indicate that it was edited from module version 1.3.0.

2.3. YANG Semantic Version Update Rules

When a new revision of a module is produced, then the following rules define how the YANG semantic version number for the new module revision is calculated, based on the changes between the two module revisions, and the YANG semantic version number of the base module revision that the changes are derived from. A two step process is used:

The first step is to classify the module change as 'editorial', 'backwards-compatible', or 'non-backwards-compatible version' using the rules defined in Section 2.1 and Section 4.

The second step is to calculate the value of the 'semver:version' field for the new module revision, based on the value of the 'semver:version' field in the base module, any how the module changes have been classified.

The following rules define how the value for the 'semver:version' argument in the new module revision is calculated:

1. If a module is being updated in a non-backwards-compatible way, then the module version "X.Y.Z[m|M]" MUST be updated to "X+1.0.0" unless that module version has already been defined with different content, in which case the module version "X.Y.Z+1M MUST be used instead.

2. If a module is being updated in a backwards-compatible way, then the next version number depends on the format of the current version number:

   i   "X.Y.Z" - the module version MUST be updated to "X.Y+1.0", unless that module version has already been defined with different content, when the module version MUST be updated to "X.Y.Z+1m instead".

   ii   "X.Y.Zm" - the module version MUST be updated to "X.Y.Z+1m".

   iii   "X.Y.ZM" - the module version MUST be updated to "X.Y.Z+1M".

3. If a module is being updated in an editorial way, then the next version number depends on the format of the current version number:

   i   "X.Y.Z" - the module version MUST be updated to "X.Y.Z+1"

   ii   "X.Y.Zm" - the module version MUST be updated to "X.Y.Z+1m".

   iii   "X.Y.ZM" - the module version MUST be updated to "X.Y.Z+1M".

4. YANG module semantic version numbers beginning with 0, i.e "0.X.Y" are regarded as beta definitions and need not follow the rules above. Either the MINOR or PATCH version numbers may be updated, regardless of whether the changes are non-backwards-compatible, backwards-compatible, or editorial.

2.4. YANG Module Semver Extension

This document defines a YANG extension to add the YANG module semantic version to a Module. The complete definition of this YANG module is in Section 9.

```
        extension module-version {
          argument semver;
        }
```

The extension would typically be used this way:

```
    module yang-module-name {

      namespace "name-space";
      prefix "prefix-name";

      import ietf-semver { prefix "semver"; }

      description
        "to be completed";

      revision 2018-02-28 {
        description "Added leaf 'wobble'";
        semver:module-version "3.1.0";
      }

      revision 2017-12-31 {
        description "Rename 'baz' to 'bar', added leaf 'wibble'";
        semver:module-version "3.0.0";
      }

      revision 2017-10-30 {
        description "Change the module structure";
        semver:module-version "2.0.0";
      }

      revision 2017-08-30 {
        description "Clarified description of 'foo-64' leaf";
        semver:module-version "1.3.1";
      }

      revision 2017-07-30 {
        description "Added leaf foo-64";
        semver:module-version "1.3.0";
      }

      revision 2017-04-20 {
        description "Add new functionality, leaf 'baz'";
        semver:module-version "1.2.0";
      }

      revision 2017-04-03 {
        description "Add new functionality, leaf 'foo'";
        semver:module-version "1.1.0";
      }

      revision 2017-04-03 {
```

```
           description "First release version.";
           semver:module-version "1.0.0";
         }

         revision 2017-01-30 {
           description "NBC changes to initial revision";
           semver:module-version "0.2.0";
         }

         revision 2017-01-26 {
           description "Initial module version";
           semver:module-version "0.1.0";
         }

         //YANG module definition starts here
```

See also "Semantic Versioning and Structure for IETF Specifications" [I-D.claise-semver] for a mechanism to combine the semantic versioning, the GitHub tools, and a potential change to the IETF process.

3. Import by Semantic Version

RFC 7950 allows YANG module 'import' statements to optionally require the imported module to have a particular revision date. In practice, importing a module with an exact revision date is overly burdensome because it requires the importing module to be updated whenever any change to the imported module occurs. The alternative choice of using an import statement without a revision date is also not ideal because the importing module may not work with all possible revisions of the imported module.

With semantic versioning, it is desirable for a importing module to specify the set of module versions of the imported module that are anticipated to be compatible.

This document specifies a YANG extension for selecting which versions of a module may be imported. It is designed around the assumption that most changes to a YANG module do not break importing modules, even if the changes themselves are not backwards compatible. E.g., fixing an incorrect pattern statement or description for a leaf would not break an import, changing the name of a leaf could break an import but frequently would not, but removing a container would break imports if it is augmented by another module.

The ietf-semver module defines the 'version' extension, a substatement to the YANG 'import' statement.

An 'import' statement MAY contain 'version' statements or a 'revision-date' statement, but not both.

The 'version' statement MAY be specified multiple times, requiring that the imported module version conforms to at least one of the 'version' statements.

The argument to the 'version' statement takes one of three valid forms:

1. "A.B.C" - import the exact module version that matches "A.B.C".

2. "A.B.C+" - import any module version that matches, or is greater than, "A.B.C".

3. "A.B.C-X.Y.Z" - import any module version that matches, or is greater than, "A.B.C"; and also matches, or is less than, "X.Y.Z". The word "MAX" can be used for 'Y' or 'Z' to represent the numerical value 32,767.

The rules for comparing module version numbers are as follows:

1. Version "R.S.T" matches version "A.B.C", only if

   R = A, S = B, and T = C

2. Version "R.S.T" is greater than version "A.B.C", only if

   R = A, S = B, and T > C; or

   R = A and S > B; or

   R > A

3. Version "R.S.T" is less than version "X.Y.Z", only if

   R = X, S = Y, and T < Z; or

   R = X and S < Y; or

   R < X

The patch modifier letter is not included as part of the 'semver:version' argument, and is entirely ignored for import statement module version number comparisons.

3.1. Module import examples

Consider an example module "example-module" that is hypothetically available in the following versions: 0.1.0, 0.2.0, 1.0.0, 1.1.0, 1.1.1m, 1.1.2M, 1.2.0, 1.2.1M, 1.2.2M, 1.3.0, 1.3.1, 2.0.0, 3.0.0, and 3.1.0. E.g. matching the versions illustrated in Section 2.2.1.

The first example selects the specific version 1.1.2M. A specific version import might be used if 1.1.2M contained changes that are incompatible with other versions.

```
import example-module {
  semver:version 1.1.2;
}
```

The next example selects module versions that match, or are greater than, version 1.2.0. This form may be used if there is a dependency on a data node introduced in version 1.2.0. This is expected to be the most commonly used form of 'import by version'.

Includes versions: 1.2.0, 1.2.1M, 1.2.2M, 1.3.0, 1.3.1, 2.0.0, 3.0.0 and 3.1.0.

```
import example-module {
  semver:version 1.2.0+;
}
```

The next example selects module versions that match, or are greater than 1.1.0, but excluding all 1.1.x and 1.2.x 'M' versions. This form may be needed if structural non backwards compatible changes are introduced in a patch 'M' version. Generally, it is advisable to avoid making such changes.

Includes versions: 1.1.0, 1.1.1m, 1.2.0, 1.3.0, 1.3.1, 2.0.0, 3.0.0, and 3.1.0.

```
import example-module {
  semver:version 1.1.0-1.1.1;
  semver:version 1.2.0;
  semver:version 1.3.0+;
}
```

The last example selects all module versions with a major version number of 1. This form may be useful if significant non backwards compatible changes have been introduced in version 2.0.0 that break import backwards compatibility.

Includes versions: 1.0.0, 1.1.0, 1.1.1m, 1.1.2M, 1.2.0, 1.2.1M, 1.2.2M, 1.3.0 and 1.3.1.

```
import example-module {
  semver:version 1.0.0-1.MAX.MAX;
}
```

4. Classifying changes in YANG modules

[RFC7950] chapter 11 defines the rules for what constitutes a backwards compatible change in YANG 1.1. However, the YANG semantic versioning scheme defined in this document uses a slightly modified version of this scheme, and also provides rules to classify changes as editorial, backwards-compatible, or non-backwards-compatible.

4.1. Editorial changes

Any changes that do not change the ordering or meaning of the YANG module in any way are classified as 'editorial'. The following rules define 'editorial':

o Changing any 'description' statement if it does not change the semantic meaning of the statement is relates to. E.g., fixing spelling or grammar, or changing layout, are all allowed.

o Adding or updating 'reference' statements.

o Adding or updating the 'organization' statement.

o Adding a new 'revision' or 'semver:module-version' statement, or correcting a previous 'revision' or 'semver:module-version' statement.

o A module may be split into a set of submodules or a submodule may be removed, provided the definitions in the module do not change except in the ways described above.

4.2. Backwards-compatible changes

[RFC7950] chapter 11 defines the rules for what constitutes a backwards-compatible change in YANG 1.1. The document update these rules in the following ways:

o Adding or changing a 'status' node to 'obsolete' is not a backwards-compatible change. Other changes/additions of status elements are backwards-compatible, as per [RFC7950].

- o Changing the ordering of statements is allowed if it does not chanage the ordering of an rpc's 'input' substatements.

4.3. Non-backwards-compatible changes

All other changes to YANG modules that are not classified as 'editorial' or 'backwards-compatible' are defined as being non-backwards-compatible.

Examples of non-backwards-compatible changes include:

- o Deleting a data node, or changing it to status obsolete.
- o Changing the name, type, or units of a data node.
- o Modifying the description in a way that changes the semantic meaning of the data node.
- o Any changes that change or reduce the allowed value set of the data node, either through changes in the type definition, or the addition or changes to 'must' statements, or changes in the description.
- o Adding or modifying 'when' statements that reduce when the data node is available in the schema.
- o Making the statement conditional on if-feature.

5. Updates to ietf-yang-library

YANG library [RFC7895] [RFC8525] is modified to support semantic versioning in three ways.

5.1. Advertising module version number

The ietf-semver YANG module augments the 'module' list in ietf-yang-library with a 'version' leaf to optionally declare the YANG semantic version of each module.

5.2. Resolving ambiguous module imports

A YANG datastore schema, defined in [RFC8525], can specify multiple revisions of a YANG module in the schema using the 'import-only' list, with the requirement from [RFC7950] that only a single revision of a YANG module may be implemented.

If a YANG module import statement does not specify a specific version or revision within the datastore schema then it could be ambiguous as

to which module revision the import statement should resolve to. Hence, a datastore schema constructed by a client using the information contained in YANG library may not exactly match the datastore schema actually used by the server.

The following rules remove the ambiguity:

If a module import statement could resolve to more than one module revision defined in the datastore schema, and one of those revisions is implemented (i.e., not an 'import-only' module), then the import statement MUST resolve to the revision of the module that is defined as being implemented by the datastore schema.

If a module import statement could resolve to more than one module revision defined in the datastore schema, and none of those revisions are implemented, but one of more modules revisions specify a YANG semantic version, then the import MUST resolve to the module with the greatest version number, according to the version comparison rules in Section 3.

If a module import statement could resolve to more than one module revision defined in the datastore schema, none of those revisions are implemented, and none of the modules revisions have a YANG semantic version number, then the import MUST resolve to the module that has the most recent revision date.

5.3. Reporting how deprecated and obsolete nodes are handled

The ietf-semver YANG module augments YANG library with two leaves to allow a server to report how it handles status 'deprecated' and status 'obsolete' nodes. The leaves are:

deprecated-nodes-implemented: If present, this leaf indicates that all schema nodes with a status 'deprecated' child statement are implemented equivalently as if they had status 'current', or otherwise deviations MUST be used to explicitly remove 'deprecated' nodes from the schema. If this leaf is absent then the behavior is unspecified.

obsolete-nodes-absent: If present, this leaf indicates that the server does not implement any status 'obsolete' nodes. If this leaf is absent then the behaviour is unspecified.

Implementations that implement the YANG semantic versioning scheme defined in this document MUST set the 'deprecated-nodes-implemented' leaf because the refined module update rules in Section 4 require that this is how servers handle 'deprecated' and 'obsolete' nodes to comply with YANG module semantic versioning.

If a server does not set the 'deprecated-nodes-implemented' leaf, then clients MUST NOT rely solely on the YANG module semantic version number to determine whether two module versions are backwards compatible, and MUST also consider whether the status of any nodes has changed to 'deprecated' and whether those nodes are implemented by the server.

6. YANG status description extension

The ietf-semver module specifies the YANG extension 'status-description' that can be used as a substatement of the status statement. The argument to this extension can contain freeform text to help readers of the module understand why the node was deprecated or made obsolete, when it is anticipated that the node will no longer be available for use, and potentially reference other schema elements that can be used instead. An example is shown below.

```
leaf imperial-temperature {
  type int64;
  units "degrees Fahrenheit";
  status deprecated {
    semver:status-description
      "Imperial measurements are being phased out in favor
       of their metric equivalents.  Use metric-temperature
       instead.";
  }
  description
    "Temperature in degrees Fahrenheit.";
}
```

7. Semantic versioning of YANG instance data

Instance data sets [I-D.ietf-netmod-yang-instance-file-format] do not have an associated YANG semantic version, as compatibility for instance data is undefined.

However, instance data may reference an associated YANG schema, and that schema could make use of semantic version numbers, both for the individual YANG modules that comprise the schema, and potentially for the entire schema itself (e.g., [I-D.rwilton-netmod-yang-packages]).

In this way, the versioning of a schema associated with an instance data set, may allow a client to determine whether the instance data could also be used in conjunction with other versions of the YANG schema, or other versions of the modules that define the schema.

One common scenario, where instance data may have to cope with changes to the schema is for the <startup> datastore when a server is

restarted with a different YANG schema (e.g. due to a software upgrade or downgrade). How a server restores the configuration from <startup> during such upgrades or downgrades is outside the scope of this specification.

8. Guidelines

8.1. Guidelines to YANG model authors

NBC changes to YANG models may cause problems to clients, who are consumers of YANG models, and SHOULD be avoided. However, there are cases where NBC changes are required, e.g. to fix an incorrect YANG model.

YANG model authors are recommended to minimize NBC changes and keep changes BC whenever possible.

The use of status "deprecated" with the status-description statement allows clients to plan a migration to alternative data nodes.

When NBC changes are introduced, consideration should be given to the impact on clients and YANG model authors SHOULD try to mitigate that impact.

8.1.1. Use of YANG semantic versioning

Module authors should use the following guidance when applying the module version update rules specified in Section 2.3.

Updates to modules SHOULD be applied to the latest version of YANG modules, avoiding the use the 'm|M' patch modifier. When used in this way, the YANG semantic version numbers are compatible with the versioning scheme defined by the semver.org 2.0.0 rules.

Changes to older versions of published YANG modules SHOULD be minimized, since there may be a greater impact on clients, and comparing between version numbers becomes more limited if the 'm|M' modifiers are used. However, if it is necessary to make such changes then the following guidelines apply:

Any changes SHOULD also be made to a new latest version of the YANG module, if appropriate.

Where possible, changes SHOULD be restricted to backwards-compatible changes only.

NBC changes MAY be made, subject to the constraints defined in Section 2.3. The impact to clients SHOULD be carefully considered and minimized if possible.

The version numbers associated with a module MUST never be reused. E.g., when updating module version 3.4.0 in a NBC manner the module author must verify whether version 4.0.0 is available for use and if that version was already used, the updated module must use version 3.4.1M instead.

Patch modifier letters (i.e. 'm' or 'M') are sticky. For example if version 3.4.1M is modified in a BC way, the next version is 3.4.2M. This is to indicate that 3.4.2M is not BC with 3.4.0, however it comes at the cost of not being able to indicate the type of change between 3.4.1M and 3.4.2M.

As explained in Appendix A.2.2, while programatically determining a semantic version is possible using tools (e.g. the pyang utility), human oversight is highly recommended because of some special cases which can not be detected by tools. Therefore, a model author SHOULD use both means to determine a model's semantic version.

8.1.2. Making non-backwards-compatible changes to a YANG module

There are various valid situations where a YANG module has to be modified in a non-backwards-compatible way. Here are the different ways in which this can be done:

o If the server can support NBC versions of the YANG module simultaneously using version selection, then the NBC changes MAY be done immediately. Clients would be required to select the version which they support and the NBC change would have no impact on them.

o When possible, NBC changes are done incrementally to provide clients time to adapt to NBC changes.

Here are some guidelines on how non-backwards compatible changes can be made incrementally:

1. The changes should be made incrementally, e.g. a data node's status SHOULD NOT be changed directly from "current" to "obsolete" (see Section 4.7 of [RFC8407]), instead the status SHOULD first be marked "deprecated" and then when support is removed its status MUST be changed to "obsolete". Instead of using the "obsolete" status, the data node MAY be removed from the model but this has the risk of breaking modules which import the modified module.

2. A node with status "deprecated" MUST be supported for the solution described here to function properly.

3. A node with status "deprecated" SHOULD be available for at least one year before its status is changed to "obsolete", see Section 4.7 of [RFC8407].

4. Support for a node which is "obsolete" is indicated by the node "obsolete-nodes-present, see Section 5.

5. The new extension "status-description" SHOULD be used for nodes which are "obsolete" or "deprecated".

6. For status "deprecated", the "status-description" SHOULD also indicate until when support for the node is guaranteed. If there is a replacement data node, rpc, action or notification for the deprecated node, this SHOULD be stated in the "status-description".

7. When obsoleting or deprecating data nodes, the "deprecated" or "obsolete" status SHOULD be applied at the highest possible level in the data tree. For example, when deprecating all data nodes in a container, the "deprecated" status SHOULD be applied to the container. For clarity, the status MAY be added in all the affected nodes but the status-description SHOULD be added only at the highest level in the tree.

The following sections have examples on how non-backwards-compatible changes can be made.

8.1.2.1. **Removing a data node**

Removing a leaf or container from the data tree, e.g. because support for the corresponding feature is being removed:

1. The node's status SHOULD be changed to "deprecated" and it MUST be supported for at least one year. This is a backwards-compatible change.

2. When the node is not available anymore, its status MUST be changed to "obsolete" and the "status-description" updated, this is a non-backwards-compatible change. The "status-description" SHOULD be used to explain why the node is not available anymore.

8.1.2.2. Changing the type of a leaf node

Changing the type of a leaf-node. e.g. consider a "vpn-id" node of type integer being changed to a string:

1. The status of node "vpn-id" SHOULD be changed to "deprecated" and the node SHOULD be available for at least one year. This is a backwards-compatible change.

2. A new node, e.g. "vpn-name", of type string is added to the same location as the existing node "vpn-id". This new node has status "current" and its description SHOULD explain that it is replacing node "vpn-id".

3. During the period of time where both nodes are available, how the server behaves when either node is set is outside the scope of this document and will vary on a case by case basis. Here are some options:

    1. A server MAY prevent the new node from being set if the old node is already set (and vice-versa). The new node MAY have a when statement to achieve this. The old node MUST NOT have a when statement since this would be a non-backwards-compatible change, but the server MAY reject the old node from being set if the new node is already set.

    2. If the new node is set and a client does a get or get-config operation on the old node, the server MAY map the value. For example, if the new node "vpn-name" has value "123" then the server MAY return integer value 123 for the old node "vpn-id". However, if the value can not be mapped, we need a way of returning "unsupported" TBD.

4. When node "vpn-id" is not available anymore, its status MUST be changed to "obsolete" and the "status-description" is updated. This is a non-backwards-compatible change.

8.1.2.3. Reducing the range of a leaf node

8.1.2.4. Changing the key of a list

8.1.2.5. Renaming a node

8.1.2.6. Changing a default value

8.2. Guidelines to YANG model clients

Guidelines for clients of modules using YANG semantic versioning:

- Clients SHOULD be liberal when processing data received from a server. For example, the server may have increased the range of an operational node causing the client to receive a value which is outside the range of the YANG model revision it was coded against

- Clients SHOULD monitor changes to published YANG modules through their version numbers, and use appropriate tooling to understand the specific changes between module versions. In particular, clients SHOULD NOT migrate to NBC versions of a module without first understanding the specifics of the NBC changes.

- Clients SHOULD plan to make changes to match published status changes. When a node's status changes from "current" to "deprecated", clients SHOULD plan to stop using that node in a timely fashion. When a node's status changes to "obsolete", clients MUST stop using that node.

9. Semantic Version Extension YANG Modules

YANG module with extensions for defining a module's YANG semantic version number, and importing by version.

```
<CODE BEGINS> file "ietf-semver@2019-02-07.yang"
module ietf-semver {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-semver";
  prefix semver;

  organization
    "IETF NETMOD (Network Modeling) Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/netmod/>
     WG List:  <mailto:netmod@ietf.org>

     Author:   Benoit Claise
               <mailto:bclaise@cisco.com>

     Author:   Joe Clarke
               <mailto:jclarke@cisco.com>

     Author:   Reshad Rahman
               <mailto:rrahman@cisco.com>

     Author:   Robert Wilton
```

```
                  <mailto:rwilton@cisco.com>

        Author:   Kevin D'Souza
                  <mailto:kd6913@att.com>

        Author:   Balazs Lengyel
                  <mailto:balazs.lengyel@ericsson.com>

        Author:   Jason Sterne
                  <mailto:jason.sterne@nokia.com>";
     description
       "This module contains a definition for a YANG 1.1 extension to
        express the semantic version of YANG modules.";

     revision 2019-02-27 {
       description
         "* Move YANG library augmentations into a separate module.
          * Update references.";
       reference
         "draft-verdt-netmod-yang-semver:
          YANG Semantic Versioning for Modules";
       semver:module-version "0.3.0";
     }

     revision 2018-04-05 {
       description
         "* Properly import ietf-yang-library.
          * Fix the name of module-semver => module-version.
          * Fix regular expression syntax.
          * Augment yang-library with booleans as to whether or not
            deprecated and obsolete nodes are present.
          * Add an extension to enable import by semantic version.
          * Add an extension status-description to track deprecated
            and obsolete reasons.
          * Fix yang-library augments to use 7895bis.";
       reference
         "draft-clacla-netmod-yang-model-update:
          New YANG Module Update Procedure";
       semver:module-version "0.2.1";
     }
     revision 2017-12-15 {
       description
         "Initial revision.";
       reference
         "draft-clacla-netmod-yang-model-update:
          New YANG Module Update Procedure";
       semver:module-version "0.1.1";
     }
```

```
     typedef version {
       type string {
         pattern '[0-9]{1,5}\.[0-9]{1,5}\.[0-9]{1,5}(m|M)?';
       }
       description
         "The type used to represent a YANG semantic version number.

         The YANG semver version number is expressed as a string of the
         form: 'X.Y.Zv'; where X, Y, and Z each represent non-negative
         integers smaller than 32768, and v represents an optional
         single character suffix: 'm' or 'M'.

         o 'X' is the MAJOR version.  Changes in the major version
           number indicate changes that are non-backwards-compatible to
           versions with a lower major version number.

         o 'Y' is the MINOR version.  Changes in the minor version
            number indicate changes that are backwards-compatible to
            versions with the same major version number, but a lower
            minor version number.

        o 'Zv' is the PATCH version and modifier.  Changes in the patch
           version number can indicate editorial, backwards-compatible,
           or non-backwards-compatible changes relative to versions with
           the same major and minor version numbers, but lower patch
           version number, depending on what form modifier 'v' takes:

           *  'M' - the change represents a non-backwards-compatible
                    change

           *  'm' - the change represents a backwards-compatible change

           * If the modifier letter is absent, the change represents an
             editorial change";

       reference
         "draft-verdt-netmod-yang-semver: YANG Semantic Versioning";
     }

     extension module-version {
       argument semver;
       description
         "The version number for the module revision it is used in.

          This format of the argument matches the type version.

          The rules for updating the module-version number are described
          in section XXX of 'YANG Semantic Versioning for Modules';
```

```
          By comparing the module-version between two revisions of a
          given module, one can determine if different revisions are
          backwards compatible or not, as well as whether or not new
          features have been added to a newer revision.

          If a module contains this extension it indicates that for this
          module the updated status and update rules as this described
          in RFC XXXX are used.

          The statement MUST only be a substatement of the 'revision'
          statements.  Zero or one module-version statement is allowed
          per parent statement. No substatements are allowed.

          'revision' statements in submodules MAY contain a
          'module-version' statement for documentation purposes, but
          its meaning is undefined, and has no effect on the including
          module's semantic version.";
       reference
         "draft-verdt-netmod-yang-semver:
          YANG Semantic Versioning for Modules";
     }

     extension import-versions {
       argument version-clause;
       description
         "This extension specifies an acceptable set of semantic
          versions of a given module that may be imported.

          The statement MUST only be a substatement of the import
          statement.

          The statement MUST NOT be present if the import has a
          revision-date substatement.

          The statement MUST NOT be present if the imported module does
          not support semantic versioning.

          Zero or more versions statements are allowed per parent
          statement.  No substatements are allowed.

          The version-clause argument MUST follow one of the below
            patterns:
              (i)  "+' \d+\.\d+\.\d+ '+"
                Matches exact version, e.g. 3.6.1

              (ii) "+ '\d+\.\d+\.\d+\+ '+"
                Matches exact version or greater, e.g. 3.6.1+
```

```
              (iii) "+' \d+\.\d+.\d+-\d+\.(\d+|MAX).(\d|MAX)+ '+"
                Matches inclusive range,
                e.g. 3.6.1-7.8.4, or 3.2.1-3.MAX.MAX";

        reference
          "draft-verdt-netmod-yang-semver: Import by Semantic Version";
      }

      extension status-description {
        argument description;
        description
          "Freeform text that describes why a given node has been
           deprecated or made obsolete.  This may point to other schema
           elements that can be used in lieu of the given node.

           This statement MUST only be used as a substatement of the
           status statement

           Zero or more status-description statements are allowed per
           parent statement.  No substatements are allowed.";
        reference
          "draft-verdt-netmod-yang-semver: YANG status description
           extension";
      }
    }
    <CODE ENDS>
```

YANG module with augmentations to YANG Library to support semantic version numbers.

```
    <CODE BEGINS> file "ietf-yl-semver@2019-02-07.yang"
    module ietf-yl-semver {
      yang-version 1.1;
      namespace "urn:ietf:params:xml:ns:yang:ietf-yl-semver";
      prefix yl-semver;

      import ietf-semver {
        prefix semver;
      }

      import ietf-yang-library {
        prefix yanglib;
      }

      organization
        "IETF NETMOD (Network Modeling) Working Group";
      contact
        "WG Web:   <https://datatracker.ietf.org/wg/netmod/>
```

```
        WG List:  <mailto:netmod@ietf.org>

        Author:   Benoit Claise
                  <mailto:bclaise@cisco.com>

        Author:   Joe Clarke
                  <mailto:jclarke@cisco.com>

        Author:   Reshad Rahman
                  <mailto:rrahman@cisco.com>

        Author:   Robert Wilton
                  <mailto:rwilton@cisco.com>

        Author:   Kevin D'Souza
                  <mailto:kd6913@att.com>

        Author:   Balazs Lengyel
                  <mailto:balazs.lengyel@ericsson.com>

        Author:   Jason Sterne
                  <mailto:jason.sterne@nokia.com>";
     description
       "This module contains augmentations to YANG Library to add module
        level semantic version numbers and to provide an indication of
        how deprecated and obsolete nodes are handled by the server.";

       semver:module-version "0.1.0";

     revision 2019-02-27 {
       description
         "Moved YANG library augmentations into a separate module.";
       reference
         "draft-verdt-netmod-yang-semver:
          YANG Semantic Versioning for Modules";
       semver:module-version "0.1.0";
     }

     augment "/yanglib:yang-library/yanglib:module-set/yanglib:module" {
       description
         "Augmentation modules with a semantic version.";
       leaf version {
         type semver:version;
         description
           "The semantic version for this module.  The version MUST
            match the semver:version value in specific revision of the
            module loaded in this module-set.";
         reference
```

```
           "draft-verdt-netmod-yang-semver: YANG Semantic Versioning";
       }
     }

     augment "/yanglib:yang-library/yanglib:schema" {
       description
         "Augmentations to the ietf-yang-library module to indicate how
          deprecated and obsoleted nodes are handled for each datastore
          schema supported by the server.";

       leaf deprecated-nodes-implemented {
         type empty;
         description
           "If present, this leaf indicates that all schema nodes with a
            status 'deprecated' child statement are implemented
            equivalently as if they had status 'current', or otherwise
            deviations MUST be used to explicitly remove 'deprecated'
            nodes from the schema.  If this leaf is absent then the
            behavior is unspecified.";
         reference
           "draft-verdt-netmod-yang-semver: Reporting how deprecated and
            obsolete nodes are handled";
       }
       leaf obsolete-nodes-absent {
         type empty;
         description
           "If present, this leaf indicates that the server does not
            implement any status 'obsolete' nodes.  If this leaf is
            absent then the behaviour is unspecified.";
         reference
           "draft-verdt-netmod-yang-semver: Reporting how deprecated and
            obsolete nodes are handled";
       }
     }
   }
   <CODE ENDS>
```

10. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The design team consists of the following members whom have worked on the YANG versioning project:

- Balazs Lengyel
- Benoit Claise
- Ebben Aries

- Jason Sterne
- Joe Clarke
- Juergen Schoenwaelder
- Mahesh Jethanandani
- Michael (Wangzitao)
- Qin Wu
- Reshad Rahman
- Rob Wilton

The initial revision of this document was refactored and built upon [I-D.clacla-netmod-yang-model-update].

Discussons on the use of Semver for YANG versioning has been held with authors of the OpenConfig YANG models. We would like thank both Anees Shaikh and Rob Shakir for their input into this problem space.

11. Security Considerations

The document does not define any new protocol or data model. There are no security impacts.

12. IANA Considerations

12.1. YANG Module Registrations

The following YANG module is requested to be registred in the "IANA Module Names" registry:

The ietf-semver module:

    Name: ietf-semver

    XML Namespace: urn:ietf:params:xml:ns:yang:ietf-semver

    Prefix: semver

    Reference: [RFCXXXX]

The ietf-yl-semver module:

    Name: ietf-yl-semver

XML Namespace: urn:ietf:params:xml:ns:yang:ietf-yl-semver

Prefix: yl-semver

Reference: [RFCXXXX]

13. References

13.1. Normative References

[I-D.verdt-netmod-yang-versioning-reqs]
Clarke, J., "YANG Module Versioning Requirements", draft-verdt-netmod-yang-versioning-reqs-02 (work in progress), November 2018.

[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <https://www.rfc-editor.org/info/rfc7895>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8407] Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

[RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <https://www.rfc-editor.org/info/rfc8525>.

13.2. Informative References

[I-D.clacla-netmod-model-catalog]
Clarke, J. and B. Claise, "YANG module for yangcatalog.org", draft-clacla-netmod-model-catalog-03 (work in progress), April 2018.

[I-D.clacla-netmod-yang-model-update]
Claise, B., Clarke, J., Lengyel, B., and K. D'Souza, "New YANG Module Update Procedure", draft-clacla-netmod-yang-model-update-06 (work in progress), July 2018.

[I-D.claise-semver]
Claise, B., Barnes, R., and J. Clarke, "Semantic Versioning and Structure for IETF Specifications", draft-claise-semver-02 (work in progress), January 2018.

[I-D.ietf-netmod-yang-instance-file-format]
Lengyel, B. and B. Claise, "YANG Instance Data File Format", draft-ietf-netmod-yang-instance-file-format-02 (work in progress), February 2019.

[I-D.openconfig-netmod-model-catalog]
Shaikh, A., Shakir, R., and K. D'Souza, "Catalog and registry for YANG models", draft-openconfig-netmod-model-catalog-02 (work in progress), March 2017.

[I-D.rwilton-netmod-yang-packages]
Wilton, R., "YANG Packages", draft-rwilton-netmod-yang-packages-00 (work in progress), December 2018.

[openconfigsemver]
"Semantic Versioning for Openconfig Models", <http://www.openconfig.net/docs/semver/>.

[semver] "Semantic Versioning 2.0.0", <https://www.semver.org>.

[yangcatalog]
"YANG Catalog", <https://yangcatalog.org>.

13.3. URIs

[1] https://github.com/netmod-wg/yang-ver-dt/issues/14

[2] https://github.com/netmod-wg/yang-ver-dt/issues/11

[3] https://github.com/netmod-wg/yang-ver-dt/issues/13

[4] https://github.com/netmod-wg/yang-ver-dt/issues/12

[5] https://github.com/netmod-wg/yang-ver-dt/issues/10

[6] https://github.com/netmod-wg/yang-ver-dt/issues/9

[7] https://github.com/netmod-wg/yang-ver-dt/issues/8

[8] https://github.com/netmod-wg/yang-ver-dt/issues/7

[9] https://github.com/netmod-wg/yang-ver-dt/issues/6

[10] https://github.com/netmod-wg/yang-ver-dt/issues/5

[11] https://github.com/netmod-wg/yang-ver-dt/issues/4

[12] https://github.com/netmod-wg/yang-ver-dt/issues/15

[13] https://github.com/netmod-wg/yang-ver-dt/issues/2

Appendix A. Appendix

A.1. Open Issues

Open issues are being tracked at <https://github.com/netmod-wg/yang-ver-dt/issues>. Currently open issues are:

- o Do we need a new version of YANG? #14 [1]
- o Add guidance text about warning NBC changes might break imports #11 [2]
- o Add a naming convention for versioned YANG file#13 [3]
- o Define editorial, bc, nbc impact of adding, changing, removing extension stmts#12 [4]
- o How to version modules in IETF drafts (after they have been published at 1.0.0 or later#10 [5]
- o The solution does not strictly support semver 2.0.0#9 [6]
- o Are whitespace changes allow between two module instances with the same version (or revision)?#8 [7]
- o Do we assume that a module has an implicit semver if none as been specified?#7 [8]
- o Is changing the ordering of nodes an NBC change?#6 [9]
- o Should version statement be at top level or under revision statement?#5 [10]
- o Figure out whether changing the imports constitute a BC or NBC change#4 [11]
- o Does BC or NBC depend on whether the node is config true/false?#15 [12]
- o Status obsolete nodes#2 [13]

A.2. Derived Semantic Version

This temporary text is intended to be moved to a separate draft the describes the tool based approach for versioning YANG modules mentioned in Section 1.2.

A.2.1. The Derived Semantic Version

If an explicitly defined semantic version is not available in the YANG module, it is possible to algoritmically calculate a derived semantic version. This can be used for modules not containing a definitive semantic-version as defined in this document or as a starting value when specifying the definitive semantic-version. Be aware that this algorithm may sometimes incorrectly classify changes between the categories non-compatible, compatible or error-correction.

A.2.2. Implementation Experience

[yangcatalog] uses the pyang utility to calculate the derived-semantic-version for all of the modules contained within the catalog. [yangcatalog] contains many revisions of the same module in order to provide its derived-semantic-version for module consumers to know what has changed between revisions of the same module.

Two distinct leafs in the YANG module [I-D.clacla-netmod-model-catalog] contain this semver notation:

o the semantic-version leaf contains the value embedded within a YANG module (if it is available).

o the derived-semantic-version leaf is established by examining the the YANG module themselves. As such derived-semantic-version only takes syntax into account as opposed to the meaning of various elements when it computes the semantic version.

o The algorithm used to produce the derived-semantic-version is as follows:

1. Order all modules of the same name by revision from oldest to newest. Include module revisions that are not available, but which are defined in the revision statements in one of the available module versions.

2. If module A, revision N+1 has failed compilation, bump its derived semantic MAJOR version. For unavailable module versions assume non-backward compatible changes were done., thus bump its derived semantic MAJOR version.

3. Else, run "pyang --check-update-from" on module A, revision N and revision N+1 to see if backward-incompatible changes exist.

4. If backward-incompatible changes exist, bump module A, revision N+1's derived MAJOR semantic version.

5. If no backward-incompatible changes exist, compare the pyang trees of module A, revision N and revision N+1.

6. If there are structural differences (e.g., new nodes), bump module A, revision N+1's derived MINOR semantic version.

7. If no structural differences exist, bump module A, revision N+1's derived PATCH semantic version.

The pyang utility checks many of the points listed in section 11 of [RFC7950] for known module incompatibilities. While this approach is a good way to programmatically obtain a semantic version number, it does not address all cases whereby a major version number might need to be increased. For example, a node may have the same name and same type, but its meaning may change from one revision of a module to another. This represents a semantic change that breaks backward compatibility, but the above algorithm would not find it. Therefore, additional, sometimes manual, rigor must be done to ensure a proper version is chosen for a given module revision.

Authors' Addresses

Benoit Claise
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Phone: +32 2 704 5622
Email: bclaise@cisco.com

Joe Clarke
Cisco Systems, Inc.
7200-12 Kit Creek Rd
Research Triangle Park, North Carolina
United States of America

Phone: +1-919-392-2867
Email: jclarke@cisco.com

Reshad Rahman
Cisco Systems, Inc.

Email: rrahman@cisco.com

Robert Wilton (editor)
Cisco Systems, Inc.

Email: rwilton@cisco.com

Balazs Lengyel
Ericsson
Magyar Tudosok Korutja
1117 Budapest
Hungary

Phone: +36-70-330-7909
Email: balazs.lengyel@ericsson.com

Jason Sterne
Nokia

Email: jason.sterne@nokia.com

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
United States of America

Email: kd6913@att.com