Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

R. Wilton
Cisco Systems, Inc.

October 22, 2018

YANG Versioning Potential Solutions
draft-verdt-netmod-yang-solutions-00

Abstract

This 'work in progress' document describes and evaluates potential solutions to the requirements stated in section 5 of the YANG versioning requirements draft. The aim of this draft is to only provide a progress update to the Netmod WG concerning the YANG versioning design team discussions on potential solutions, and to hopefully provide minimally sufficient information to allow the wider Netmod community to provide input into the direction of the YANG versioning design team.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2019.

Copyright Notice

Table of Contents

1. Terminology and Conventions 2
2. Introduction 3
3. Background 3
4. Summary of requirements 4
5. Potential solutions to core YANG versioning requirements . . 5
5.1. Module level 'major.minor.patch' semantic versioning . . 6
5.2. Module level 'major.minor.patch(x)' modified semantic versioning 7
5.3. Module level 'release.major.minor.patch' partial semantic versioning 9
5.4. A tool based approach comparing YANG schema modules/trees 10
5.5. Follow existing RFC 7950 rules 11
6. Solutions to related YANG versioning issues 12
7. Open Questions 12
7.1. Is YANG module revision date preserved? 13
7.2. Do YANG update rules allow for bug fixes? 13
7.3. Does one size fit all? 13
7.4. Should vendors we allowed to version YANG modules as part of a release train? 13
7.5. How should versioning apply to submodules? 14
7.6. Is having a patch version number useful for YANG modules? 14
8. Contributors 14
9. Security Considerations 15
10. IANA Considerations 15
11. References 15
11.1. Normative References 15
11.2. Informative References 15
Author's Address 15

1. Terminology and Conventions

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].

This document also makes use of the terminology introduced in the YANG versioning requirements draft (REF REQUIRED). In addition, this document introduces the following terminology:

- bc: Used as an abbreviation for a backwards-compatible change.

- nbc: Used as an abbreviation for a non-backwards-compatible change.

- editorial change: A backwards-compatible change that does not change the YANG module semantics in any way.

2. Introduction

This draft represents transient work in progress, and should be read as such. In particular, the descriptions of the solutions are not intended to be complete, nor necessarily consider all scenarios, but instead are intended to explore the broad approach and key aspects of the particular solution. The solution descriptions do not address all requirements at this time, instead they focus on the requirements that have the most significance on the final direction of the solution. Nor does this draft recommend any particular solution or solutions at this time. It is anticipated that once a final solution approach has been decided upon, that a separate draft shall be produced that will supersede this temporary draft.

The remainder of this document is split into the following sections:

Chapter Section 4 provides a condensed summary of the requirements, taken from [I-D.verdt-netmod-yang-versioning-reqs]. This section also lists where in the document these requirements are considered, if at all.

A significant part of this document is aimed at discussing the potential 'core' solutions, which are focussed on solving requirements: R1.1, R1.2, R1.4, R2.1, R2.2 and R4.4, and described in chapter Section 5.

Possible solutions for some of the secondary requirements, such as datanode lifecycle management, are considered in chapter Section 6. In particular, possible solutions for requirements R1.3, R4.1, R4.2 and R4.3 are considered.

Finally, chapter Section 7 lists some of the open issues that the YANG versioning design team are considering and working through. For some questions, a tentative design team direction of the answer is also given.

3. Background

Some members of the design team are authors of a potential solution draft to the YANG versioning requirements. The purpose of this document is to ensure that all reasonable solutions to the YANG

versioning problem have been properly considered before converging on a single chosen solution.

4. Summary of requirements

The requirement themselves are documented in section 5 of XXX. A shortened, non normative, summary of each of the requirements (using the same requirement numbers) is provided below to aid evaluation of the potential solutions.

Req 1.1 - MUST support nbc updates without breaking imports.

Req 1.2 - MUST support nbc updates without breaking existing client code.

Req 1.3 - MUST support import stmt restricted to only some revisions.

Req 1.4 - MUST support modules to be versioned by software release.

Req 2.1 - MUST be able to determine if two arbitrary versions of any MODULE are unchanged, bc, or nbc.

Req 2.2 - SHOULD be able to determine if two arbitrary versions of any DATA NODE are unchanged, bc, or nbc.

Req 3.1 - MUST allow servers to support existing clients.

Req 3.2 - MUST allow for simultaneously support of clients using different (perhaps restricted) revisions.

Req 4.1 - MUST provide way to indicate if deprecated nodes are implemented.

Req 4.2 - MUST be able to document reason for lifecycle changes, and possible alternative data nodes.

Req 4.3 - MUST be able to forewarn of future lifecycle changes.

Req 4.4 - SHOULD allow fixes to older revision of a module.

Req 5.1 - MUST provide guidance on how to use the new scheme.

Req 5.2 - MUST provide, and document, an upgrade path from existing YANG/protocols.

Req 5.3 - MUST consider versioning impact on instance data.

The following list indicates where solutions for particular requirements are considered in this draft.

Req 1.1 - Section 5, core solutions

Req 1.2 - Section 5, core solutions

Req 1.3 - Section 6, extra solutions

Req 1.4 - Section 5, core solutions

Req 2.1 - Section 5, core solutions

Req 2.2 - Section 5, core solutions

Req 3.1 - Deferred until main solution direction is chosen.

Req 3.2 - Deferred until main solution direction is chosen.

Req 4.1 - Section 6, extra solutions

Req 4.2 - Section 6, extra solutions

Req 4.3 - Section 6, extra solutions

Req 4.4 - Section 5, core solutions

Req 5.1 - Deferred until main solution direction is chosen.

Req 5.2 - Deferred untli main solution direction is chosen.

Req 5.3 - Deferred until main solution direction is chosen.

5. Potential solutions to core YANG versioning requirements

This section considers solutions that are aimed at solving the main YANG versioning requirements. In particular, the solutions described here are aimed at solving the following requirements: R1.1, R1.2, R1.4, R2.1, R2.2 and R4.4.

The solutions being considered are:

1. Module level 'major.minor.patch' semantic versioning

2. Module level 'major.minor.patch(x)' modified semantic versioning

3. Module level 'release.major.minor.patch' versioning

4. A tool based approach comparing YANG schema modules/trees

5. Follow existing RFC 7950 rules

5.1. Module level 'major.minor.patch' semantic versioning

This solution introduces a module level version number that adopts a subset of the semantic versioning rules published at semver.org.

The key part of this solution is a version number that comprises three fields, 'major.minor.patch':

1. major - updated only when a non-backwards-compatible change is made

2. minor - updated only when a backwards-compatible change is made

3. patch - updated only for 'editorial' changes that do not change the API semantics in any way

When a field in the version number is incremented, all following fields are reset back to 0. Major version number 0 indicates that the module is not yet stable and allows non-backwards-compatible changes without requiring the major version number to be incremented (e.g., this could be used in IETF drafts before they become RFCs).

If this solution is adopted, it is assumed that vendors would need to manage versioning of vendor YANG models independently of software release trains, and even then they would be limited in the scope of what changes are possible in an already shipped release, which is anticipated to not meet the business requirements of some vendors.

Solution advantages:

1. Follows widely known semantic versioning rules.

2. Version number alone indicates whether 2 module revisions are backwards-compatible.

3. Sufficient for most (but not necessarily all) YANG models developed by SDOs.

4. Matches the scheme being used by OpenConfig YANG models.

Solution disadvantages:

1. Does not fully support long lived vendor software release trains. In particular:

Does not necessarily allow for backwards-compatible changes (enhancements or fixes) in older releases.

Does not allow for non-backwards-compatible changes (enhancements or fixes) in older releases.

2. The 'patch' field is not as useful for YANG modules (which act like an API), since 'editorial' changes are likely to be less common that backwards-compatible enhancements and fixes.

5.2. Module level 'major.minor.patch(x)' modified semantic versioning

This solution modifies the semantic versioning solution described previously, with the principal aim of allow fixes to released code.

The change to the semantic versioning solution is a modification to how the 'patch' field is used. In addition to 'editorial' changes that do not change the YANG module semantics, the patch field can also be used in a limited way to indicate major and minor version changes as well. If the patch field is incremented for a minor version change that it is appended with the suffix '(m)', if the patch field is incremented for a major version change then it is appended with the suffix '(M)', replacing '(m)', if present. Once a given 'major.minor' version has a patch field value with '(m)' or '(M)' then all subsequent patch revisions on the same 'major.minor' version retain the letter '(m)' or '(M)' regardless of whether the subsequent changes are backwards-compatible, non-backwards-compatible, or editorial changes.

The updated semantic versioning rules for updating the 'major.minor.patch' version number is as follows:

1. if a non-backwards-compatible change is made then either the major version number MUST be updated (resetting the minor and patch version numbers to 0) or only the patch version number MUST be updated and appended with '(M)', replacing '(m)' if present.

2. if a backwards-compatible change is made then either the minor version number MUST be updated (resetting the patch version numbers to 0) or only the patch version number MUST be updated and appended with '(m)' unless the previous patch version number already had '(M)' appended, in which case the '(M)' suffix is retained for the new patch version.

3. if an editorial change is made then the patch version number MUST be updated. If the previous patch version number already had either an '(m') or '(M)' suffix then it is retained for the new patch version.

When a field in the version number is incremented, all following fields are reset back to 0. Major version number 0 indicates that the module is not yet stable and allows non-backwards-compatible changes without requiring the major version number to be incremented (e.g., this could be used in IETF drafts before they become RFCs).

If this solution is adopted, it is assumed that vendors would need to manage versioning of vendor YANG models independently of software release trains, but that they are able to release fixes to bugs in YANG module versions that are present in long lived software releases.

Where possible, the version number should be updated using the standard semantic versioning rules, relying on the '(m)' and '(M)' suffixes only used where strictly necessary.

Solution advantages:

1. Allows fixes to released YANG modules, whilst still preserving semver like semantics.

2. Aims to be sufficient for SDO and vendor YANG modules.

3. Modules can choose to just use semver rules if they wish. E.g. the scheme is compatible with the scheme being used by OpenConfig YANG models.

Solution disadvantages:

1. Slightly more complex than standard semver.org rules. The (m|M) suffix may be confusing, and their significance misinterpreted.

2. Within a 'major.minor' version branch it is not possible to determine whether a specific change is backwards-compatible or not.

3. If on a version with the (m) suffix, e.g. 'A.B.C(m)', it is not possible to determine whether an update to 'A.D.E', where D > B is a backwards-compatible change.

Variants:

Rather than using '(m)' or '(M)', it could instead use separate counters for bc and nbc changes, facilitating meaningful semantic versioning comparison between different patch versions on 'major.minor' branch.

Rather than overloading the patch version number, separate semantic version numbers could be used on branches. E.g. if a bc fix was required to version '1.2.3' this could be presented as '1.2.3/1.1.0', if there was a further nbc fix then the next branch version would be '1.2.3/2.0.0'.

5.3. Module level 'release.major.minor.patch' partial semantic versioning

This solution extends the semver 'major.minor.patch' version number scheme, by prefixing it with an explicit software release positive integer field.

The key part of this solution is a version number comprising four fields (release.major.minor.patch):

1. release - may be updated at any time (e.g. for a new major software release)

2. major - updated only when a non-backwards-compatible change is made

3. minor - updated only when a backwards-compatible change is made

4. patch - updated only only for changes that do not change the API semantics in any way

When a field in the version number is incremented, all following fields are reset back to 0, except for major that resets to 1. Release version number 0 indicates that the version is not yet stable and non-backwards-compatible changes are allowed without incrementing the major version number.

The assumption for this scheme is that the release number is always incremented for every major release, i.e. at any point where nbc changes may potentially be required in an older release.

Solution advantages:

1. Supports long lived vendor software release trains.

2. Completely allows bc and nbc changes (enhancements or fixes) in older independent releases.

3. Probably sufficient for YANG models developed by both vendors and SDOs.

Solution disadvantages:

1. Release version field must be incremented regardless of changes.

2. Version number is no longer an indicator of changes between 2 module revisions. I.e. the main benefit of semantic versioning is lost.

3. Differs from the scheme used by OpenConfig YANG model.

Similar variants:

The 'release' field could be regarded as optional, and if omitted, the version interpreted in exactly the same way as the module level 'major.minor.patch' semantic versioning solution.

5.4. A tool based approach comparing YANG schema modules/trees

This solution relies on using tooling to compare either two YANG modules, or two YANG schema trees to identify any changes between the two modules that do not conform to RFC 7950 section 11 backwards-compatibility rules.

Not all differences between two YANG statements in different module versions can easily be identified as backwards-compatible or not (for example changes in description, pattern statements, must or when statements may be hard to check). If a tool is unable to check then it would have to flag the change as potentially being non-backwards-compatible, potentially reporting many false positives.

To mitigate this, it is proposed that this solution also introduces a new YANG extension statement to indicate that a change is backwards-compatible.

When comparing a module schema, a tool would also be able to take into account enabled features, deviations, and the subset of the schema being used by the client. This would allow a tooling based approach to give a more accurate answer as to whether a client would be affected when upgrading between two software versions.

Solution advantages:

1. Gives the most accurate answer that works in all cases.

Solution disadvantages:

1. Cannot easily check whether two modules are compatible just by looking at them. Probably needs to be used in conjunction with a module level versioning scheme.

2. Differs from the scheme used by OpenConfig YANG models.

5.5. Follow existing RFC 7950 rules

The final choice is to decide that the existing mechanism described in RFC 7950, that disallows any non-backwards-compatible changes in a given model, is the best way forward. Instead of making a nbc chagne, the modeller can introduce new parallel nodes, and deprecate the existing nodes within the same module. Alternatively an entirely new module, with a separate name and namespace can be introduced.

As a solution, this cannot meet all of the requirements stated in the requirements draft.

If this solution was sufficient, then the YANG versioning design team would not have been formed. However, some vendors are pragmatically ignoring the strict YANG module update rules (e.g. for vendor modules).

Solution advantages:

1. No significant change in YANG language semantics required. Changes, or perhaps extensions, could be made to the YANG language to address some of other requirements that have independent solutions.

Solution disadvantages:

1. If an nbc has to be made (even for a minor feature) then there is a high impact to all clients using the module, servers implementing the module, and other YANG modules that import from the module. This impact would be particularly acute for a core YANG module that is being updated in an nbc way, that is imported by many other YANG modules. Hence, choosing this solution really means that there can be no nbc changes to a module unless the module is being restructured in a major way when a separate name for the module makes sense regardless.

2. Seems to make standardization slow because participants are seemingly try harder to get the perfect model first because the cost of having to change it seems so high.

3. Old, dead definitions can potentially never be removed from a module.

4. Does not work well for vendor generated YANG models, since they cannot easily have the level or control and stability required for it to never change.

5. Does not solve the problem where deviations are used to introduce nbc changes.

6. Introduces a problem where a single underlying property is represented by two (or more) independent data nodes in the same schema. There does not appear to be a clean solution on how to manage the relationship between these two nodes (e.g. if both an old and new client are interacting with a server). Other solutions have the potential of handling this better.

Variants:

One variant of this solution is to agree on the rules for making fixes to published YANG modules, and determine whether that requires any changes to the section 11 text in RFC 950.

6. Solutions to related YANG versioning issues

These partial solutions address particular point requirements. The partial solutions are:

1. Deprecated flag - Add a flag to YANG library to indicate whether deprecated nodes are implemented or not. This is a potential solution to Req 4.1.

2. Redefine deprecated stmt - Change the definition of the YANG deprecated statement to indicate that deprecated data nodes must be implemented, or otherwise deviated. This is a potential solution to Req 4.1.

3. Status description - Allow the "description" statement under the YANG "status" statement to document data node lifecycle, and allow for forward guidance. This is a potential solution to Reqs 4.2 and 4.3.

4. Alternative node path - Introduce a new YANG statement to provide an alternative path for a deprecated, or obsolete, data node. This is a potential additional solution to Req 4.2 and perhaps also Req 4.3.

7. Open Questions

This section lists some of the open questions that the design team is still grappling with.

7.1. Is YANG module revision date preserved?

With the introduction of the new versioning scheme, should every YANG module still have a revision statement, or is that entirely superseded by a new version statement? Is it required that YANG modules revision dates MUST be unique for different versions of a module?

The position that the DT is tending towards is:

> All revision dates for YANG modules must be unique. The slight complexity of requiring this should minimize the impact to existing tooling.
>
> it is acceptable to break the existing monotonically increasing property of the current module revision date, but within a given 'stream' of YANG modules the monotonically increasing property should be preserved.

7.2. Do YANG update rules allow for bug fixes?

Does YANG (RFC 7950) section 11 allow nbc fixes to existing models, and if so, are there any limits as to what form those fixes can take, or are these strictly prohibited by the module update rules?

7.3. Does one size fit all?

Potentially different types of YANG modules may want to follow different versioning semantics.

E.g. it may be right that standardized YANG modules are very slow changing and conservative in their backwards compatibility

Conversely, it is potentially more pragmatic that vendor YANG modules need to change in more significant ways mirroring changes in underlying implementations or hardware.

7.4. Should vendors we allowed to version YANG modules as part of a release train?

Some of the solutions described in this document probably require vendors to version vendor YANG modules outside of release trains, which is likely to be different to how some vendors are managing this today. Is it a reasonable constraint to put on vendors that they MUST version YANG modules outside of a release train to provide a cleaner version history?

7.5. How should versioning apply to submodules?

Submodules can have different revision dates from the including parent module. Does this mean that submodules should be versioned independently of their parent module? Or should the version number apply only at the module level?

Need to consider the upgrade rules allow definitions to be moved between submodules.

7.6. Is having a patch version number useful for YANG modules?

The semantic versioning solution on semver.org is designed to version both APIs and implementations. In this scenario, the patch level versioning number is particularly useful to indicate a fix in the implementation, where the API has not changed. The versioning for YANG modules is primarily concerned with the API semantics rather than implementation, and hence the patch level version number is not so directly useful, where its purpose is limited to changes that do not affect semantics of the YANG module (e.g. fixes to typos for example).

8. Contributors

This document grew out of the YANG module versioning design team that started after IETF 101. The following people are members of that design team and have contributed to defining the problem and specifying the requirements:

- Balazs Lengyel
- Benoit Claise
- Ebben Aries
- Jason Sterne
- Joe Clarke
- Juergen Schoenwaelder
- Mahesh Jethanandani
- Michael (Wangzitao)
- Qin Wu
- Reshad Rahman

- Rob Wilton
- Susan Hares

9. Security Considerations

The document does not define any new protocol or data model. There is no security impact.

10. IANA Considerations

None

11. References

11.1. Normative References

[I-D.verdt-netmod-yang-versioning-reqs] Clarke, J., "YANG Module Versioning Requirements", draft-verdt-netmod-yang-versioning-reqs-01 (work in progress), October 2018.

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

[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>.

11.2. Informative References

[RFC8049] Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8049, DOI 10.17487/RFC8049, February 2017, <https://www.rfc-editor.org/info/rfc8049>.

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

[RFC8299] Wu, Q., Ed., Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8299, DOI 10.17487/RFC8299, January 2018, <https://www.rfc-editor.org/info/rfc8299>.

Author's Address

Robert Wilton
Cisco Systems, Inc.