Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: March 6, 2016

R. Wilton
Cisco Systems

September 3, 2015

"With-config-state" Capability for NETCONF/RESTCONF

draft-wilton-netmod-opstate-yang-00

Abstract

This document proposes a possible alternative solution for handling applied configuration state in YANG as described in draft-openconfig-netmod-opstate-01. The proposed solution, roughly modelled on the with-defaults NETCONF/RESTCONF capability, aims to meet the key requirements set out in draft-openconfig-netmod-opstate-01 without the need for YANG module authors to explicitly duplicate configuration nodes in both configuration and operational containers. This draft does not address the issue of co-location of configuration and operational state for interfaces, nor does it provide a NETCONF mechanism to retrieve operational data separately from configuration data.

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 http://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 March 6, 2016.

Table of Contents

1. Introduction

The Consistent Modeling of Operational State Data Internet Draft [I-D.openconfig-netmod-opstate] sets out a number of operational requirements and proposed solutions for handling intended and applied config state when using YANG models. This document sets out a possible alternative solution for some of those requirements.

The solution proposed in this document does not require any changes to any existing YANG modules to support intended and applied config state. In particular: the proposed solution does not require the data models to be explicitly modelled with separate configuration and operational containers, and it does not require that all configuration and operational state nodes and leaves to be defined as groupings.

Nor does the proposed solution make explicit use of separate datastores to model intended configuration separately from applied configuration.

Instead, the solution proposed here is a method for generating an enhanced schema based on any YANG model that is optionally used by network management protocols. This enhanced schema includes up to four data leaves for each configuration node defined in the YANG model. These cover both the intended and applied values, along with an additional reason code and message if the applied configuration does not match the intended configuration.

Although the solution described here is only defined in the context of NETCONF and RESTCONF, it should be possible to extend the same YANG config data encoding mechanism to other protocol schemes used to access YANG data if required.

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 RFC 2119 [RFC2119].

The following terms are defined in Consistent Modeling of Operational State Data Internet Draft [I-D.openconfig-netmod-opstate], and reproduced here for convenience.

intended configuration - this data represents the state that the network operator intends the system to be in. This data is colloquially referred to as the 'configuration' of the system.

applied configuration - this data represets the state that the network element is actually in, i.e. that which is currently being run by particular software modules (e.g. the BGP daemon), or other systems within the device (e.g. a secondary control-plane, or line card).

derived state - this data represents information which is generated as part of the system's own interactions. For example, derived state may consist of the results of protocol interactions (such as the negotiated duplex state of an Ethernet link), statistics (such as message queue depth), or counters (such as packet input or output bytes).

The following additional terms are used in this document:

operational nodes - this term is colloquially used in this draft to refer to "config false" YANG nodes.

2. Objectives

The aim of this draft is to provide a partial alternative solution to the requirements set out in Consistent Modeling of Operational State Data Internet Draft [I-D.openconfig-netmod-opstate]. An explanation of how the specific requirements are addressed is described in Section 7.

3. "With-config-state" encoding scheme

The solution proposed in this document makes use of a new encoding scheme that is used to represent YANG configuration nodes in NETCONF and RESTCONF. An optional parameter, called <with-config-state> and defined below, indicates when this new encoding scheme is used.

When the with-config-state option is used each YANG configuration leaf in the datastore is returned in a different format. Rather than being encoded as an XML or JSON leaf element that holds the configured value, it is instead returned as a node element, with the same name as the YANG config leaf, that contains up to four separate leaf elements. The four leaf elements that the node may contain are 'cfg-intended', 'cfg-applied', 'cfg-status', and 'cfg-status-reason'. Theses leaves are externally addressable through using the full path of the leaf, providing explicit distinct paths for intended configuration vs applied configuration. These elements are described in more detail in the following sub-sections. Concrete examples of the encoding for NETCONF and RESTCONF requests are given in Appendix A.

3.1. cfg-intended

The cfg-intended leaf represents the intended configuration of the device, and is of the same datatype and holds the same value as the normal YANG data model configuration leaf. The cfg-intended leaf is only present if the associated configuration node exists in the YANG data model.

The cfg-intended leaf is semantically equivalent to the config leaf in the YANG data model that is based on, and hence is logically read/writable. In particular, when the <with-config-state> parameter is used, management requests to modify the configuration may also use the full path to the cfg-intended leaf. The server semantics for writing to the cfg-intended leaf are exactly the same as for writing to the standard YANG config node path - the flexibility is provided as a convenience to the NMS client.

3.2. cfg-applied

The cfg-applied leaf represents the applied configuration, and is of the same datatype as a normal YANG data model configuration leaf. If there is no configuration currently in effect then the cfg-applied leaf is not present.

The cfg-applied leaf is read only.

To give some examples:

If the configuration has been successfully applied then the cfg-applied leaf would exactly match the cfg-intended leaf.

If a new item of configuration is in the process of being applied then the cfg-intended leaf holds the intended configuration value,

and the cfg-applied leaf would not be present until the configuration is in effect.

If an existing item of configuration is in the process of being deleted then the cfg-applied leaf would hold the current configuration value, and the cfg-intended leaf would not be present. Once the delete operation has completed, the configuration node element itself would logically be deleted.

If the configuration value of an existing item of configuration is in the process of being changed, then the cfg-intended leaf would hold the new proposed value, and the cfg-applied leaf would hold the existing value that is currently in effect.

3.3. cfg-status

The cfg-status leaf is used, when required, to indicate why the value of the cfg-applied leaf does not match the value of the cfg-intended leaf. It is only present when the values of the cfg-intended and cfg-applied leaves do not match.

The cfg-status leaf is read only.

The cfg-status leaf can take one of following values:

in-progress - the config operation is in the process of being applied.

waiting - the config operation is waiting for other configuration to be applied or hardware to be available before it can be applied. Additional specific information may be provided in the cfg-status-reason leaf.

failed - the config operation failed to be applied. Additional information may be provided in the cfg-status-reason leaf to indicate the reason for the failure.

3.4. cfg-status-reason

The cfg-status-reason leaf may be used to provide additional information as to why the value of the cfg-applied leaf does not match the value of the cfg-intended leaf.

The cfg-status-reason leaf may only be present in the case that the cfg-status leaf is present and is set to either waiting or failed.

The cfg-status-reason leaf is read only.

3.5. Non-leaf config nodes

Non-leaf config nodes require some special handling. In particular, containers with presence and list elements must be considered.

The proposed solution for both types of node is the same. The cfg-intended, cfg-applied, cfg-status, and cfg-status-reason leaf nodes are implicitly added as direct descendants of the presence-container or list element.

Note: There is an open issue that using these leaves directly opens up a potential naming clash between the "cfg-*" names above and existing explicitly defined child nodes in the YANG module definition. There are a few possible ways that this might be addressed:

Making the four "cfg-*" leaves reserved names. I.e. to ensure that they are not used in general YANG modules.

By inserting an implicit node between all child nodes under the container or list element. This would automatically ensure that there can be no naming clash between the defined YANG nodes and the implicitly added "cfg-*" leaves.

By using a reserved namespace for the "cfg-*" leaves to ensure that they cannot clash with any explicitly defined in the YANG module.

4. Retrieval of intended and applied configuration

To make use of the new encoding scheme defined above, this document defines a new parameter, called <with-config-state>, which can be added to specific NETCONF operation request messages, or as a RESTCONF query parameter, to control how retrieval of configuration nodes is treated by the server.

The <with-config-state> parameter is supported for the following NETCONF operations: <get>, <get-config>, <edit-config>, <delete-config>.

The <with-config-state> query parameter is supported for the following RESTCONF operations: GET, PUT, POST, PATCH, DELETE.

Use of the <with-config-state> parameter ensures that all config nodes are always returned using the defined encoding. It also allows servers to explicitly reference the cfg-* leaves in requests and updates.

A server that implements this specification MUST accept the <with-config-state> parameter containing the enumeration for any of the with-config-state modes it supports. The <with-config-state> parameter contains one of the four enumerated values defined in this section.

4.1. all-cfg

When data is retrieved with a <with-config-state> parameter equal to 'all-cfg', all 'cfg-*' nodes are reported using the encoding scheme defined in Section 3.

4.2. intended-cfg-only

When data is retrieved with a <with-config-state> parameter equal to 'intended-cfg', only the 'cfg-intended' leaves are reported using the encoding scheme defined in Section 3. All other 'cfg-*' leaves are omitted.

4.3. applied-cfg-only

When data is retrieved with a <with-config-state> parameter equal to 'applied-cfg-only', only the 'cfg-applied' leaves are reported using the encoding scheme defined in Section 3. All other 'cfg-*' leaves are omitted.

4.4. diff-cfg-only

When data is retrieved with a <with-config-state> parameter equal to 'diff-cfg-only', config nodes are only returned if the value of the cfg-intended leaf does not match the value of the cfg-applied leaf. If the config node is returned then all appropriate 'cfg-*' leaves are returned as per the encoding scheme defined in Section 3.

5. "With-config-state" Capability

5.1. Overview

The :with-config-state capability indicates whether a server supports the with-config-state functionality. For a server that indicates support for the :with-config-state capability it must support at least the 'all-cfg' option. It may also indicate support for the additional with-config-state retrieval modes.

5.2. Dependencies

None

5.3. Capability Identifier

urn:ietf:params:netconf:capability:with-config-state:1.0

The identifer has a paramater: "also-supported". This parameter indicates which additional enumeration values (besides 'all-cfg') the server will accept for the <with-config-state> parameter in Section 4. The value of the parameter is a comma-separated list of one or more modes that are supported. Possible modes are 'intended-cfg-only', 'applied-cfg-only', 'diff-cfg-only' as defined in Section 4.

6. Suggested layout of data models

Generally, to ensure that operational data and configuration data can be easily related, this draft recommends that configuration and associated operational nodes be co-located in the same YANG container. More precisely, YANG clients should be able to assume that configuration and operational nodes within the same container are implicitly related.

7. Addressing the requirements of the Consistent Modeling of Operational State Data draft

7.1. Addressing requirement 3: 'To interact with both intended and applied configuration'

The proposed solution in this draft provides a way for a NMS to explicitly access both the intended and applied configuration state of configuration nodes. It also provides a convenient way that both the intended and applied configuration values can be returned and easily compared. It also has the following additional benefits:

It optionally provides additional information as to why the applied configuration does not match the intended configuration.

It does not force the YANG modules to use groupings for configuration data so that it can be mirrored in the operational state. In particular, it places no burden to support an eventual consistency configuration model on YANG modules that do not need to operate in that environment.

The <with-config-state> parameter in the extension allows the client to request that only configuration nodes that are not in the intended state are returned.

This draft also addresses the issue of allowing a NMS to easily relate configuration and operational state. As should be clear the relationship between cfg-intended and cfg-applied states for a particular node are trivially and efficiently mappable for all YANG configuration nodes. With the exception of interface operational state, that is not addressed by this draft, the relationship between configuration and derived state is acheived through the convention that co-located configuration and operational state be held in the same YANG container. This is semantically similar to the approach in Consistent Modeling of Operational State Data Internet Draft [I-D.openconfig-netmod-opstate] that implicitly binds the contents of the 'config' and 'state' YANG container nodes together if they are rooted to the same parent YANG container.

7.2. Addressing requirement 4.1: Applied config as part of operational state

This requirement is met through the use of the separate cfg-intended and cfg-applied implict leaf nodes that are available when using the <with-config-state> extension parameter set to 'intended-cfg-only' or 'applied-cfg-only' with either the NETCONF <get-config> operation or the RESTCONF GET request with the 'content' query parameter set to 'config'.

7.3. Addressing requirement 4.2: Support for both transactional, synchronous management systems as well as distributed, asynchronous management systems

Devices that only support a transactional sychronous management system have the choice of either not supporting the <with-config-state> extension, or alternatively may achieve compliance with this extension fairly easily by returning the same value for both cfg-intended and cfg-applied leaf nodes, and always omitting the cfg-status and cfg-status-reason leaves. Any requests using the path to the cfg-intended and cfg-applied leaves can be mapped back to the base config leaf defined in the YANG data model. Any explicit requests get or get-config requests for cfg-status and cfg-status-reason can be rejected.

Devices that support an asynchronous configuration system would implement support for the extension and provide the cfg-* leaves defined in this draft when requested.

7.4. Addressing requirement 4.3: Separation of configuration and operational state data; ability to retrieve them independently

The first point is addressed by the proposed solution. Config and operational data are already split, and the naming of the cfg-intended vs cfg-applied leaves provides a clear distinction between intended configuration, applied configuration, and derived state.

The second point is not fully addressed by this draft. The proposed protocol extension allows for just the intended config vs applied config nodes to be returned. RESTCONF already supports querying config separately from operational state through use of the 'content' query paremeter. A separate NETCONF protocol extension would be required to return just the operational nodes without any of the configuration nodes, such as the <get-state> enhancement described in Operational State Enhancements for YANG, NETCONF, and RESTCONF [I-D.kwatsen-netmod-opstate].

7.5. Addressing requirement 4.4: Ability to retreive operational state corresponding only to derived values, statistics, etc

Not directly addressed by this draft. RESTCONF already supports querying config separately from operational state through use of the 'content' query paremeter. A separate NETCONF protocol extension would be required to return just the operational nodes without any of the configuration nodes, such as the <get-state> enhancement described in Operational State Enhancements for YANG, NETCONF, and RESTCONF [I-D.kwatsen-netmod-opstate].

7.6. Addressing requirement 4.5: Consistent schema locations for configuration and corresponding operational state data

Section 4.5 of Consistent Modeling of Operational State Data Internet Draft [I-D.openconfig-netmod-opstate] indicates that it is desirable to have a well defined path to retrieve the cfg-intended vs cfg-applied values to avoid requiring external context when referencing that information. This is achieved by allowing paths in the NETCONF and RESTCONF protocols to include one of the cfg-state leaves when using the <with-config-state> extension. E.g. if the path to a particular config leaf was normally /..path-to-leaf../cfg-leaf then the intended config value could be referenced and obtained by using /..path-to-leaf../cfg-leaf/cfg-intended. The cfg-applied, cfg-status, and cfg-status-reason leaves can all be referenced and accessed in a similar fashion.

Containers with presence are not leaf nodes, and hence require slightly differently handling to configuration leaf nodes. The proposed solution is that containers with presence contain the cfg-

intended, cfg-applied, cfg-status, and cfg-status-reason leaf nodes as direct descendants of the container node and hence can be accessed using the same scheme as for config leaves. E.g. if the path to a particular container with presence was normally /..path-to-p-container../cfg-p-container/ then the intended config value could be referenced and obtained by using /..path-to-p-container../cfg-p-container/cfg-intended. The cfg-applied, cfg-status, and cfg-status-reason leaves can all be referenced and accessed in a similar fashion.

8. Acknowledgements

The authors wish to thank Einar Nilsen-Nygaard, Neil Ketley, Peyman Owladi for their helpful comments, ideas and expertise contributing to this draft.

9. IANA Considerations

TBD. This document would at least need to register a new capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry for the with-config-state optional capability.'".

10. Security Considerations

The proposal in this document does not have any security considerations beyond the existing NETCONF/RESTCONF/YANG security considerations.

11. References

11.1. Normative References

[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.openconfig-netmod-opstate]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01 (work in progress), July 2015.

[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>.

[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>.

[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>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <http://www.rfc-editor.org/info/rfc6243>.

11.2. Informative References

[I-D.kwatsen-netmod-opstate] Watsen, K., Bierman, A., Bjorklund, M., and J. Schoenwaelder, "Operational State Enhancements for YANG, NETCONF, and RESTCONF", draft-kwatsen-netmod-opstate-00 (work in progress), September 2015.

Appendix A. Encoding examples for NETCONF and RESTCONF

A sample encoding of the <with-config-state> enhancement is described below.

A simple example module is provided to illustrate the subsequent examples. This is not a real module, and is not intended for any real use.

```
module example {

namespace "http://example.com/ns/interfaces";

prefix exam;

container interfaces {
  description "Example interfaces group";

  list interface {
    description "Example interface entry";
    key name;

    leaf name {
      description
        "The administrative name of the interface.";
      type string {
        length "1 .. max";
      }
    }

    leaf mtu {
      description
        "The maximum transmission unit (MTU) value assigned to
         this interface.";
      type uint32;
      default 1514;
    }
  }
}
```

A.1. NETCONF get-config request using with-config-state with all-cfg option

A get-config request is made for the interfaces subtree using the <with-config-state> enhancement and 'all-cfg' option that returns all config nodes with explicit cfg-intended and cfg-applied leaves, and cfg-status and cfg-status-reason leaves when appropriate.

In this example, at the time of processing the get-config request, the NETCONF server is also asynchronously processing a request to set the MTU leaf to 9000 for 4 interface config nodes.

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-config-state
     xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-config-state">
      all-cfg
    </with-config-state>
  </get>
</rpc>
```

The response indicates that at the time of the reply:

The request to set the MTU leaf on eth0/0 to 9000 has completed.

The request to change the MTU leaf on eth0/1 from 2000 to 9000 is in progress.

The request to set the MTU leaf on eth0/2 to 9000 is in progress.

The request to set the MTU leaf on eth1/0 to 9000 is blocked because the necessary hardware is not present.

```
<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="http://example.com/ns/interfaces">
      <interface>
        <cfg-intended/>
        <cfg-actual/>
        <name>
          <cfg-intended>eth0/0</cfg-intended>
          <cfg-actual>eth0/0</cfg-actual>
        </name>
        <mtu>
          <cfg-intended>9000</cfg-intended>
          <cfg-actual>9000</cfg-actual>
        </mtu>
      </interface>
      <interface>
        <cfg-intended/>
        <cfg-actual/>
        <name>
          <cfg-intended>eth0/1</cfg-intended>
          <cfg-actual>eth0/1</cfg-actual>
        </name>
```

```
          <mtu>
            <cfg-intended>9000</cfg-intended>
            <cfg-actual>2000</cfg-actual>
            <cfg-status>in-progress</cfg-status>
          </mtu>
        </interface>
        <interface>
          <cfg-intended/>
          <cfg-actual/>
          <name>
            <cfg-intended>eth0/2</cfg-intended>
            <cfg-actual>eth0/2</cfg-actual>
          </name>
          <mtu>
            <cfg-intended>9000</cfg-intended>
            <cfg-status>in-progress</cfg-status>
          </mtu>
        </interface>
        <interface>
          <cfg-intended/>
          <cfg-status>waiting</cfg-status>
          <cfg-status-reason>Linecard 1 is not available
          </cfg-status-reason>
          <name>
            <cfg-intended>eth1/0</cfg-intended>
            <cfg-status>waiting</cfg-status>
            <cfg-status-reason>Linecard 1 is not available
            </cfg-status-reason>
          </name>
          <mtu>
            <cfg-intended>9000</cfg-intended>
            <cfg-status>waiting</cfg-status>
            <cfg-status-reason>Linecard 1 is not available
            </cfg-status-reason>
          </mtu>
        </interface>
      </interfaces>
    </data>
  </rpc-reply>
```

A.2. NETCONF get-config request using with-config-state with diff-cfg-only option

A get-config request is made for the interfaces subtree using the <with-config-state> enhancement and 'diff-cfg-only' option that only returns nodes where the cfg-intended node does not match the cfg-applied node. Appropriate parent nodes are also returned.

As per the previous examples, at the time of processing the get-config request, the NETCONF server is also asynchronously processing a request to set the MTU leaf to 9000 for 4 interface config nodes.

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-config-state
     xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-config-state">
      diff-cfg-only
    </with-config-state>
  </get>
</rpc>
```

The response indicates that the outstanding configuration requests still to be processed are:

The request to change the MTU leaf on eth0/1 from 2000 to 9000 is in progress.

The request to set the MTU leaf on eth0/2 to 9000 is in progress.

The request to set the MTU leaf on eth1/0 to 9000 is blocked because the necessary hardware is not present.

```
<rpc-reply message-id="102"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="http://example.com/ns/interfaces">
      <interface>
        <cfg-intended/>
        <cfg-actual/>
        <name>
          <cfg-intended>eth0/1</cfg-intended>
          <cfg-actual>eth0/1</cfg-actual>
        </name>
        <mtu>
          <cfg-intended>9000</cfg-intended>
          <cfg-actual>2000</cfg-actual>
          <cfg-status>in-progress</cfg-status>
        </mtu>
      </interface>
      <interface>
        <cfg-intended/>
        <cfg-actual/>
```

```
          <name>
            <cfg-intended>eth0/2</cfg-intended>
            <cfg-actual>eth0/2</cfg-actual>
          </name>
          <mtu>
            <cfg-intended>9000</cfg-intended>
            <cfg-status>in-progress</cfg-status>
          </mtu>
        </interface>
        <interface>
          <cfg-intended/>
          <cfg-status>waiting</cfg-status>
          <cfg-status-reason>Linecard 1 is not available
          </cfg-status-reason>
          <name>
            <cfg-intended>eth1/0</cfg-intended>
            <cfg-status>waiting</cfg-status>
            <cfg-status-reason>Linecard 1 is not available
            </cfg-status-reason>
          </name>
          <mtu>
            <cfg-intended>9000</cfg-intended>
            <cfg-status>waiting</cfg-status>
            <cfg-status-reason>Linecard 1 is not available
            </cfg-status-reason>
          </mtu>
        </interface>
      </interfaces>
    </data>
  </rpc-reply>
```

A.3. NETCONF get-config request using with-config-state with applied-cfg-only option

A get-config request is made for the interfaces subtree using the <with-config-state> enhancement and 'applied-cfg-only' option that only returns the currently applied configuration.

As per the previous examples, At the time of processing the get-config request, the NETCONF server is also asynchronously processing a request to set the MTU leaf to 9000 for 4 interface config nodes.

```
<rpc message-id="103"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <filter type="subtree">
      <interfaces xmlns="http://example.com/ns/interfaces"/>
    </filter>
    <with-config-state
     xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-with-config-state">
      applied-cfg-only
    </with-config-state>
  </get>
</rpc>
```

The response indicates that the current applied configuration of the selected nodes is:

The MTU leaf of eth0/0 is 9000.

The MTU leaf of eth0/1 is 2000.

Eth0/2 has no MTU leaf applied.

[Implicitly - there is no applied configuration for Eth1/0 since the hardware is not present.]

```
<rpc-reply message-id="103"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <interfaces xmlns="http://example.com/ns/interfaces">
      <interface>
        <cfg-actual/>
        <name>
          <cfg-actual>eth0/0</cfg-actual>
        </name>
        <mtu>
          <cfg-actual>9000</cfg-actual>
        </mtu>
      </interface>
      <interface>
        <cfg-actual/>
        <name>
          <cfg-actual>eth0/1</cfg-actual>
        </name>
        <mtu>
          <cfg-actual>2000</cfg-actual>
        </mtu>
      </interface>
      <interface>
        <cfg-actual/>
        <name>
          <cfg-actual>eth0/2</cfg-actual>
        </name>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

A.4. RESTCONF GET request using with-config-state with all-cfg option (JSON)

An equivalent RESTCONF/JSON example to Appendix A.1 is provided to illustrate the equivalent JSON encoding.

A REST GET request is made for all config data using the <with-config-state> enhancement and 'all-cfg' option that all returns all config nodes with explicit cfg-intended and cfg-applied leaves.

In this example, at the time of processing the GET request, the RESTCONF server is also asynchronously processing a request to set the MTU leaf to 9000 for 4 interface config nodes.

```
GET /restconf/data/example-events:events?content=config&with-config-state=all-
cfg
  HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

As per Appendix A.1, the response indicates that at the time of the reply:

The request to set the MTU leaf on eth0/0 to 9000 has completed.

The request to change the MTU leaf on eth0/1 from 2000 to 9000 is in progress.

The request to set the MTU leaf on eth0/2 to 9000 is in progress.

The request to set the MTU leaf on eth1/0 to 9000 is blocked because the necessary hardware is not present.

```
HTTP/1.1 200 OK
Date: Mon, 1 Apr 2015 04:01:00 GMT
Server: example-server
Content-Type: application/yang.data+json

{
"example:interfaces": [
  {
    "cfg-intended" = null,
    "cfg-actual" = null,
    "name" : {
      "cfg-intended" = "eth0/0",
      "cfg-actual" = "eth0/0"
    },
    "mtu" : {
      "cfg-intended" = 9000,
      "cfg-actual" = 9000
    },
  },
  {
    "cfg-intended" = null,
    "cfg-actual" = null,
    "name" : {
      "cfg-intended" = "eth0/1",
      "cfg-actual" = "eth0/1"
    },
    "mtu" : {
      "cfg-intended" = 9000,
      "cfg-actual" = 2000,
```

```
          "cfg-status" = "in-progress"
```

```
      },
    },
    {
      "cfg-intended" = null,
      "cfg-actual" = null,
      "name" : {
        "cfg-intended" = "eth0/2",
        "cfg-actual" = "eth0/2"
      },
      "mtu" : {
        "cfg-intended" = 9000,
        "cfg-status" = "in-progress"
      },
    },
    {
      "cfg-intended" = null,
      "cfg-status" = "waiting",
      "cfg-status-reason" = "Linecard 1 is not available",
      "name" : {
        "cfg-intended" = "eth1/0",
        "cfg-status" = "waiting",
        "cfg-status-reason" = "Linecard 1 is not available",
      },
      "mtu" : {
        "cfg-intended" = 9000,
        "cfg-status" = "waiting",
        "cfg-status-reason" = "Linecard 1 is not available",
      },
    },
  ]
  }
```

Author's Address

Robert Wilton
Cisco Systems

Email: rwilton@cisco.com