Operational State Enhancements for YANG, NETCONF, and RESTCONF
draft-kwatsen-netmod-opstate-02

Abstract

This document presents enhancements to YANG, NETCONF, and RESTCONF satisfying the requirements set forth in Terminology and Requirements for Enhanced Handling of Operational State.

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 August 5, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ........................................... 2
2. Terminology ........................................... 2
3. Conceptual Model ....................................... 3
4. Enhancements to YANG ................................. 4
   4.1. The related-state Statement ...................... 4
   4.2. YANG Module ....................................... 5
5. Enhancements to NETCONF .............................. 7
   5.1. The ‘applied’ <source> Value ..................... 7
   5.2. The <sync-behavior> Parameter ..................... 7
   5.3. The <get-diff> Operation .......................... 9
   5.4. The <get-state> Operation ......................... 10
   5.5. The <sync-complete> Notification ................. 11
   5.6. YANG Module ....................................... 11
6. Enhancements to RESTCONF ............................ 23
7. Security Considerations .............................. 23
8. IANA Considerations ................................. 23
9. Acknowledgements ..................................... 23
10. References ........................................... 23
    10.1. Normative References ............................. 23
    10.2. Informative References ......................... 24
Appendix A. Traceability Matrix ........................ 25
Authors’ Addresses ..................................... 27

1. Introduction

This document presents enhancements to YANG [RFC6020], NETCONF [RFC6241], and RESTCONF [draft-ietf-netconf-restconf] satisfying the requirements set forth in Terminology and Requirements for Enhanced Handling of Operational State [draft-ietf-netmod-opstate-reqs-04].

A traceability matrix illustrating how each requirement is satisfied is provided in Appendix A.

2. 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 [RFC6241]:

- client
The following terms are defined in [draft-ietf-netmod-opstate-reqs-04]:

- applied configuration
- asynchronous configuration operation
- derived state
- intended configuration
- operational state
- synchronous configuration operation

3. Conceptual Model

The following diagram illustrates the conceptual model presented in this document:

```
+--------+    |    +--------+
| intended|    |    | applied |
| state   |    |    | config  |
+--------+    |    +--------+

+-------------------------+
| config true             |
| YANG nodes              |
+-------------------------+
    | intended |
    | config   |
    +--------+

+-------------------------+
| config false            |
| YANG nodes              |
    | derived |
    | state   |
    +--------+
```

Key aspects illustrated in the above diagram include:

- The horizontal line partitions what is defined by config true and config false nodes. Above the line are data models defined by config true nodes and below the line are data models defined by config false nodes. Notably, it illustrates that the same config
true nodes define the data model for both the intended
configuration and the applied configuration, consistent with
requirement 1-C in [draft-ietf-netmod-opstate-reqs-04].

- The vertical line partitions the types of data models in a server.
  Left of the line is data that represents the intended state of the
  server and right of the line data that represents the operational
  state of the server. Notably, the intended state of the server
  consists only of config true nodes, whereas the operational state
  consists of both config true and config false nodes. This diagram
  illustrates that operational state is composed of both applied
  config and derived state, consistent with its terminology
definition in [draft-ietf-netmod-opstate-reqs-04].

4. Enhancements to YANG

To support the opstate requirements, one modification to YANG is
necessary. This modification can be introduced as a YANG extension,
as described next.

4.1. The related-state Statement

The "related-state" statement designates where related derived state
nodes for a config true node can be found. This association is not
needed for any descendant config false nodes, as they are already
implicitly associated to the parent config true node.

The "related-state" statement takes as an argument a string that is
used to specify the path to a config false node holding the
associated operational state. The format of the argument is the same
as for the leafref’s "path" statement, Section 9.9.2 in [RFC6020].

4.1.1. Usage Example

The following example illustrates the related-state statement in use:
module ex-interfaces {
    namespace "http://example.com/interfaces";
    prefix xif;

    import ietf-yang-related-state {
        prefix yrs;
    }

    container interfaces {
        list interface {
            key name;
            yrs:related-state
                "/interfaces-state/interface[name=current()]/name";
            leaf name { type string }
            leaf mtu { type uint16; }
            ...
        }
    }

    container interfaces-state {
        config false;
        list interface {
            key name;
            leaf name { type string; }
            ...
        }
    }
}

4.2. YANG Module

Note: there is no tree diagram for this YANG module since it does not contain any protocol accessible nodes.

<CODE BEGINS> file "ietf-yang-related-state@2016-02-02.yang"

module ietf-yang-related-state {
    prefix yrs;

    organization
        "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

    contact
        "WG Web:  <http://tools.ietf.org/wg/netmod/>
        WG List:  <netmod@ietf.org>"
description
 "This YANG module defines the YANG statement 'related-state'.";

revision 2016-02-02 {
  description
   "Initial revision";
  reference
   "RFC XXXXX: Terminology and Requirements for Enhanced Handling
     of Operational State";
}

extension related-state {
  argument path;
  description
   "The related-state statement is used to identify a node that
    contains additional operational state associated for a config
    true node.

    The format of the argument is the same as for a leafref’s
    'path' statement.

    The related-state statement can be specified in the following
    YANG statements:

    o leaf
    o leaf-list
    o container
    o list

    The related-state statement allows the following YANG
    substatements:

    o description

    Multiple related-state statements can be given in a
    specific node.";
}

}
5. Enhancements to NETCONF

The following sections describe enhancements to NETCONF provided by the "ietf-netconf-opstate" YANG module.

5.1. The 'applied' <source> Value

The "ietf-netconf-opstate" YANG module enables NETCONF clients to access the contents of the applied configuration datastore by passing the value ‘applied’ for the <source> parameter in the <get-config> and <copy-config> operations.

5.1.1. Usage Example

To retrieve the "/interfaces" subtree from the applied configuration datastore:

```xml
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
        <source>
            <applied xmlns="urn:example:ietf-netconf-applied-config"/>
        </source>
        <filter type="subtree">
            <interfaces xmlns="http://example.com/example-app"/>
        </filter>
    </get-config>
</rpc>
```

5.2. The <sync-behavior> Parameter

The "ietf-netconf-opstate" YANG module enables NETCONF clients to control if operations that affect the intended configuration are executed synchronously or asynchronously. The operations that support this ability include:

- <edit-config> // only when target is "running"
- <copy-config> // only when target is "running"
- <commit>

In order to control how an operation is executed, clients must pass the <sync-behavior> parameter with a value ‘sync’ or ‘async’, for synchronous or asynchronous execution respectively.

When synchronous execution is requested (i.e., ‘sync’ is passed), the <rpc-reply> will not return until the server has fully attempted to
update both the intended and applied configurations. When no errors or warnings are generated, the <rpc-reply> will only contain <ok>. When only warnings are generated, the <rpc-reply> will contain <ok> and also the <apply-warnings> flag. If any errors are generated, the <rpc-reply> will contain the standard <rpc-error>.

When asynchronous execution is requested (i.e., ‘async’ is passed), the <rpc-reply> will return immediately, and contain an additional <sync-id> value that clients can use to correlate a subsequent ‘sync-complete’ notification (Section 5.5) to determine when the asynchronous request has completed and with what result.

For either synchronous or asynchronous requests, any processing semantics associated with the operation are preserved and extended to also include applying the configuration to internal server components. Notably are the <edit-config> operation’s <error-option> parameter and the <commit> operation’s atomic update semantic. Any rollback that may occur will restore both the intended and the applied configurations to their previous states.

In order to implement the rollback behavior for <edit-config> or <commit>, it is necessary for the server to maintain a global lock until the processing is complete. That is, either a ‘sync’ request returns or an ‘async’ request’s ‘sync-complete’ notification has been sent. Any attempts to read or write either intended or applied configuration will be blocked until the request completes.

5.2.1. Usage Example

To edit the "/interfaces" subtree from the applied configuration datastore:
5.3. The <get-diff> Operation

The "ietf-netconf-opstate" YANG module enables NETCONF clients to request a server to return the difference between any two datastores (e.g., running, startup, candidate, and applied).

The <get-diff> RPC takes two 'source' parameters as input and, based on the value of the 'format' parameter, returns either an <edit-config> document (Section 7.2 in [RFC6241]) or a YANG-patch document ([draft-ietf-netconf-yang-patch]), that transforms the first datastore into the second datastore.

5.3.1. Usage Example

This example assumes a YANG module has a data model like:

```
module example-module {
    namespace "http://example.com/ns/example-module";
    container system {
        leaf hostname {
            type string;
        }
    }
}
```
With a value in the candidate datastore of 'us-east-dc-fw-1' and value in the running datastore 'us-eass-dc-fw-1' (notice the typo):

```xml
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-diff>
    <format>yang-patch</format>
    <source1>candidate</source1>
    <source2>running</source2>
  </get-diff>
</rpc>
```

```xml
<rpc-reply message-id="101"
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-patch">
    <patch-id>202</patch-id>
    <edit>
      <edit-id>edit1</edit-id>
      <operation>replace</operation>
      <target>/example-module:system/hostname</target>
      <value>
        <hostname xmlns="http://example.com/ns/example-module">
          us-east-dc-fw-1
        </hostname>
      </value>
    </edit>
  </yang-patch>
</rpc-reply>
```

5.4. The <get-state> Operation

The "ietf-netconf-opstate" YANG module enables NETCONF clients to request a server to return operational state data. This RPC is similar to the <get-config> RPC (Section 7.1 in [RFC6241]), but for operational state instead of configuration.

The <get-state> RPC takes two optional parameters. The first parameter is a choice between the values <applied/> and <derived/> , which selects if only these types of nodes should be returned. The second parameter is a filter just like the one used by the <get-config> RPC, including the optional support for XPath expressions.

5.4.1. Usage Example

The following example depicts a request that returns all the derived state (i.e. config false) nodes under the "users" subtree.
5.5. The <sync-complete> Notification

The <sync-complete> notification notifies clients when an asynchronous request has completed. This notification includes the ‘sync-id’ value that was returned to the client that originally executed the asynchronous request, as well as any errors or warnings that the request may have been generated while processing the request.

5.5.1. Usage Example

The following example depicts a notification indicating that sync-id "4123" completed with warnings.

```xml
<notification
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-opstate">
  <eventTime>2007-07-08T00:02:00Z</eventTime>
  <sync-complete
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-opstate">
    <sync-id>4123</sync-id>
    <rpc-warning/>
  </sync-complete>
</notification>
```

5.6. YANG Module

The tree diagram for the "ietf-netconf-opstate" YANG module:

```yaml
module: ietf-netconf-opstate
augment /nc:get-config/nc:input/nc:source/nc:config-source:
  +--rw applied?   empty
augment /nc:copy-config/nc:input/nc:source/nc:config-source:
  +--rw applied?   empty
augment /nc:edit-config/nc:input:
  +--- sync-behavior? enumeration
augment /nc:edit-config/nc:output:
```

```xml
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-state>
    <derived/>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.2/config">
        <users/>
      </top>
    </filter>
  </get-state>
</rpc>
```
The "ietf-netconf-opstate" YANG module:
<CODE BEGINS> file "ietf-netconf-opstate@2016-02-02.yang"

module ietf-netconf-opstate {
    yang-version 1.1;
    prefix ncos;
    import ietf-netconf { // RFC 6241
        prefix nc;
    }
    import ietf-yang-patch { // RFC YYYY
        prefix yp;
    }
    organization "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
    contact "WG Web: <http://tools.ietf.org/wg/netmod/>
    WG List: <netmod@ietf.org>
    WG Chair: Tom Nadeau <tnadeau@brocade.com>
    WG Chair: Kent Watsen <kwatsen@juniper.net>
    WG Chair: Juergen Schoenwaelder <j.schoenwaelder@jacobs-university.de>";
    description "This YANG module does the following:

    - enables get-config and copy-config to access applied datastore
    - enables edit-config, copy-config, and commit to be executed
      either synchronously or asynchronously
    - defines 'get-diff' operation to diff datastores
    - defines 'get-state' operation to return just state data
    - defines 'sync-complete' notification for async requests";

    revision 2016-02-02 {
        description "Initial revision";
        reference "RFC XXXXX: Terminology and Requirements for Enhanced Handling
                     of Operational State";
    }
}

    // features

    feature sync {
        description "Indicates the server supports the 'sync' value for the
<sync-behavior> parameter in the operations <edit-config>,
<copy-config>, and <commit>.

```
feature async {

description
  "Indicates the server supports the 'async' value for the
<sync-behavior> parameter in the operations <edit-config>,
<copy-config>, and <commit>."
}
```

// The following two augmentations enable 'applied' to be passed as a
// <source> argument in the <get-config> and <copy-config> operations.

augment /nc:get-config/nc:input/nc:source/nc:config-source {

description
  "Allows 'applied' to be passed in <get-config>";

leaf applied {

type empty;

description
  "Indicates that the applied configuration should be returned";
}
}

augment /nc:copy-config/nc:input/nc:source/nc:config-source {

description
  "Allows 'applied' to be passed in <copy-config>";

leaf applied {

type empty;

description
  "Indicates that the applied configuration should be returned";
}
}

// The following two groupings define input and output parameters that
// are subsequently augmented into the edit-config, copy-config, and
// commit operations.

grouping sync-behavior-input {

description
  "This grouping is augmented into the edit-config, copy-config,
  and commit operations";

leaf sync-behavior {

type enumeration {
  enum "sync" {
    if-feature sync;

    description
  }
}
```
"Request the server to not return a response until both the intended and applied configurations have been updated."

} enum "async" {
  if-feature async;
  description
  "Request the server to not wait for the configuration to be applied before returning a response. The response will contain the 'sync-id' leaf. Later, after both the intended and applied configurations have been updated, the server will send the 'sync-complete' notification using the same sync-id value."
}
}

description
"This value specifies whether the server should process the request synchronously or asynchronously."

}
}

grouping sync-behavior-output {
  description
  "This grouping is augmented into the edit-config, copy-config, and commit operations. The additional output is only returned when 'sync-behavior' is specified, and thus does not alter legacy behavior."

  choice request-type {
    description
    "This choice makes it explicit which content is possibly returned for the two 'sync-behavior' values."

    case sync {
      leaf apply-warning {
        type empty;
        description
        "Indicates that a warning was generated when applying the intended configuration to internal server components (e.g., due to missing hardware) and hence the applied configuration differs from the intended configuration.";
      }
    }

    case async {
      leaf sync-id {
        type string;
        description
        "A server-specified value to identify the asynchronous request. This value is returned whenever 'async' is passed in the request. The server must ensure that it is a unique value over some reasonable amount of time.";
      }
    }
  }
}
// The following six augmentations are used to insert the
// sync-behavior-input and sync-behavior-output nodes into
// the edit-config, copy-config, and commit operations.

augment /nc:edit-config/nc:input {
    description
    "Allows ‘sync-behavior’ to be passed in <edit-config>. When specified, the ‘error-option’ value is interpreted as spanning both the updating of the intended and applied configurations. Notably, when rollback-on-error is set, both the intended and applied configurations are restored to their initial states on error."
    uses sync-behavior-input {
        when "nc:target/nc:running" {
            description "only available when target is ‘running’";
        }
    }
}

augment /nc:edit-config/nc:output {
    description
    "Allows ‘apply-warning’ and ‘sync-id’ to be returned for <edit-config> operations."
    uses sync-behavior-output;
}

augment /nc:copy-config/nc:input {
    description
    "Allows ‘sync-behavior’ to be passed in <copy-config>"
    uses sync-behavior-input {
        when "nc:target/nc:running" {
            description "only available when target is ‘running’";
        }
    }
}

augment /nc:copy-config/nc:output {
    description
    "Allows ‘apply-warning’ and ‘sync-id’ to be returned for <copy-config> operations."
    uses sync-behavior-output;
}
augment /nc:commit/nc:input {
    description
    "Allows ‘sync-behavior’ to be passed in the <commit> operation. The <commit> operation’s atomic behavior is extended to include both the updating of the intended and applied configurations.";
    uses sync-behavior-input;
}

augment /nc:commit/nc:output {
    description
    "Allows ‘apply-warning’ and ‘sync-id’ to be returned for <commit> operations.";
    uses sync-behavior-output;
}

// The following grouping is used by the get-diff RPC

grouping get-diff-source {
    description
    "This grouping is used by the ‘get-diff’ RPC’s two input parameters.";
    choice config-source {
        mandatory true;
        description
        "The configuration datastore to use as a source.";
        leaf candidate {
            if-feature nc:candidate;
            type empty;
            description
            "The candidate configuration is the config source.";
        }
        leaf running {
            type empty;
            description
            "The running configuration is the config source.";
        }
        leaf startup {
            if-feature nc:startup;
            type empty;
            description
            "The startup configuration is the config source.";
        }
        leaf applied {
            type empty;
            description
            "The applied configuration is the config source.";
        }
    }
}
The following RPC returns the diff between two datastores:

```yard
rpc get-diff {
  description
  "Get the difference between two datastores in the form of a YANG Patch (see RFC YYYY) that transforms 'source1' into 'source2'. Specifying the same datastore for both sources always returns an empty diff."
  input {
    leaf format {
      type enumeration {
        enum edit-config { description ""; } 
        enum yang-patch { description ""; } 
      }
      description "Selects output format";
    }
    container source1 {
      description "the first of two datastores to compare";
      uses get-diff-source;
    }
    container source2 {
      description "the second of two datastores to compare";
      uses get-diff-source;
    }
  }
  output {
    choice format {
      case edit-config {
        anyxml data {description "";}
      }
      case yang-patch {
        uses yp:yang-patch;
      }
      description "";
    }
  }
}
```

The following RPC returns just operational state nodes from the server:

```yard
rpc get-state {
  description
  "Retrieve operational state from the server. Operational state is composed of both applied configuration and derived state.
```

```yaml```
Applied configuration is the subset of config true nodes that is operational. Derived state is composed of just the config false nodes.;
reference "RFC 6241, Section 7.7";

input {
  choice type {
    description
    "An optional parameter to select if only applied configuration or derived state nodes should be returned. If not specified, then all node types are returned.";
    leaf applied {
      type empty;
      description
      "Only return matching applied configuration nodes";
    }
    leaf derived {
      type empty;
      description
      "Only return matching derived state nodes";
    }
  }
}

anyxml filter {
  mandatory true;
  description
  "This parameter specifies the portion of the applied configuration and derived state data to retrieve."
  nc:get-filter-element-attributes;
}

output {
  anyxml data {
    description
    "Copy of the running datastore subset and/or state data that matched the filter criteria (if any). An empty data container indicates that the request did not produce any results.";
  }
}

// This following notification is used to alert clients when an asynchronous operation request has completed.

notification sync-complete {
  description
  "Sent by the server when an asynchronous operation request has completed.";
leaf sync-id {
    type string;
    mandatory true;
    description
        "The server-specified unique identifier returned to the client that requested the asynchronous operation.";
}
choice response {
    mandatory true;
    leaf ok {
        type empty;
        description
            "Indicates that the request processed successfully.";
    }
    leaf apply-warning {
        type empty;
        description
            "Indicates that a warning was generated when applying the intended configuration to internal server components (e.g., due to missing hardware) and hence the applied configuration differs from the intended configuration.";
    }
    container rpc-error {
        leaf error-type {
            type enumeration {
                enum transport { description "transport"; }  
                enum rpc { description "rpc"; }  
                enum protocol { description "protocol"; }  
                enum application { description "application"; }  
            }
            mandatory true;
            description "error-type";
        }
        leaf error-tag {
            type nc:error-tag-type;
            mandatory true;
            description "error-tag";
        }
        leaf error-severity {
            type nc:error-severity-type;
            mandatory true;
            description "error-severity";
        }
        leaf error-app-tag {
            type string;
            description "error-app-tag";
        }
    }
}
leaf error-path {
  type string;
  description "error-path";
}
leaf error-message {
  type string;
  description "error-message";
}
container error-info {
  leaf session-id {
    type uint32;
    description "session-id";
  }
  leaf bad-attribute {
    type string;
    description "bad-attribute";
  }
  leaf bad-element {
    type string;
    description "bad-element";
  }
  leaf ok-element {
    type string;
    description "ok-element";
  }
  leaf err-element {
    type string;
    description "err-element";
  }
  leaf noop-element {
    type string;
    description "noop-element";
  }
  leaf bad-namespace {
    type string;
    description "bad-namespace";
  }
  description "error-info";
  description "rpc-error";
  description "response";
}

<CODE ENDS>
6. Enhancements to RESTCONF

TBD

7. Security Considerations

TBD

8. IANA Considerations

TBD

9. Acknowledgements

TBD

10. References

10.1. Normative References

[draft-ietf-netconf-restconf]

[draft-ietf-netconf-yang-patch]


10.2. Informative References

[draft-ietf-netmod-opstate-reqs-04]
Appendix A. Traceability Matrix

This section explains how the requirements specified in [draft-ietf-netmod-opstate-reqs-04] are satisfied by the solution presented in this document.

The following outline mimics the outline in the requirements draft:

1. Ability to interact with both intended and applied configuration
   
   A. A NETCONF client can ask the operational components of a server (e.g., line cards) for the configuration that they are currently using either by using the <get-config> RPC with the <source> value "applied", or by using the <get-state> RPC with the <applied> parameter.
   
   B. A NETCONF client is only able to read applied configuration. Neither the <get-config> nor <get-state> RPCs enable modification and no other RPC targets the applied datastore directly.
   
   C. The data model for the intended and applied configurations are identical. A fundamental aspect of the solution defined in this document is that config true nodes simultaneously define both
   
   D. Ensuring that the applied configuration values match the intended configuration values is the responsibility of an implementation more so than the solution presented in this document.

2. Support for both synchronous and asynchronous configuration operations (see terms)
   
   A. A server may only support synchronous configuration operations by only advertising the ‘sync’ feature. A server may only support asynchronous configuration operations by only advertising the ‘async’ feature. A server may support both synchronous and asynchronous configuration operations by advertising both the ‘sync’ and ‘async’ features. A NETCONF client may select on a per-operation basis if a request should be processed synchronously and asynchronously using the <sync-behavior> parameter.
   
   B. NETCONF clients can use the <get-diff> RPC to determine the difference between the intended and applied configurations.
C. This solution handles errors by extending existing semantics for the <edit-config>, <get-config>, and <commit> operations to include also the application of the configuration to the operational components of the system. For synchronous operations, errors or warnings are returned in the <rpc-reply>, whereas for asynchronous operations, errors or warnings are returned in the <sync-complete> notification. This solution supports the rollback-on-error semantics in <edit-config> (for servers that support the :rollback capability) and in <commit> (for servers that support the :candidate capability).

3. Separation of the applied configuration and derived state aspects of operational state; ability to retrieve them independently and together
   
   A. A NETCONF client can retrieve only the applied configuration of operational state either by using the "applied" source target in the <get-config> operation or by using the 'applied' argument in the <get-state> operation.
   
   B. A NETCONF client can retrieve only the derived state aspects of operational state by using the 'derived' argument in the <get-state> operation.
   
   C. A NETCONF client may retrieve both the applied configuration and derived state aspects of operational state together by not passing either the 'applied' or 'derived' arguments in the <get-state> operation.

4. Ability to relate configuration with its corresponding operational state
   
   A. Mapping intended config nodes to corresponding applied config nodes is automatic, as the same paths are used to access the same nodes in both trees.
   
   B. The ability to relate intended config nodes to associated derived state nodes is provided by the "related-state" YANG statement (Section 4.1).
   
   C. The "related-state" statement is programmatically consumable, being defined using a YANG statement.

5. Backwards compatibility
   
   A. NETCONF and RESTCONF servers can be upgraded to one that supports this solution without breaking backwards
compatibility as all the changes made to NETCONF and RESTCONF require a client to explicitly opt into it, by passing some new input parameter in the requests that it may send to a server.

B. NETCONF and RESTCONF clients coded to support this solution can differentiate servers that support opstate from those that don’t, by examining if the servers support the ietf-netconf-opstate or ietf-restconf-opstate modules.

Authors’ Addresses

Kent Watsen
Juniper Networks
EMail: kwatsen@juniper.net

Andy Bierman
Yumaworks
EMail: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems
EMail: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University Bremen
EMail: j.schoenwaelder@jacobs-university.de