

NETCONF Working Group
Internet-Draft
Updates: 8040 (if approved)
Intended status: Standards Track
Expires: January 4, 2018

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
July 3, 2017

RESTCONF Update to Support the NMDA
draft-dsdt-netconf-restconf-nmda-00

Abstract

This document updates RESTCONF [RFC8040] in order to support the Network Management Datastore Architecture (NMDA) defined in [I-D.ietf-netmod-revised-datastores].

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- o "XXXX" --> the assigned RFC value for this draft

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- o "2017-07-03" --> the publication date of this draft

The following two Appendix sections are to be removed prior to publication:

- o Appendix A. Change Log

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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 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. Requirements Language	3
3. Summary of Updates to RFC 8040	3
4. Conformance	3
5. The {+restconf}/ds/<datastore> Resource	4
6. Protocol Operations	4
7. Security Considerations	4
8. IANA Considerations	4
9. Normative References	4
Authors' Addresses	5

1. Introduction

This document updates RESTCONF [RFC8040] in order to support the Network Management Datastore Architecture (NMDA) defined in [I-D.ietf-netmod-revised-datastores].

The solution presented in this document is backwards compatible with [RFC8040]. This is achieved by it only adding new top-level

resources, and thereby leaving the semantics of all existing resources alone.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Summary of Updates to RFC 8040

This document updates [RFC8040] in the following ways:

- o Adds new top-level resource `/ds`.
- o Add new query parameter `'with-origin'`.
- o Section 3.5.4, Paragraph 3 doesn't apply for `<operational>`.

4. Conformance

A RESTCONF server identifies that it supports NMDA both by supporting the `<operational>` datastore, as well as by supporting at least revision `YANG_LIBRARY_REVISION` of the `"ietf-yang-library"` module, as specified in [I-D.nmdsd-netconf-rfc7895bis].

RESTCONF clients MAY test if a server supports NMDA using the HEAD method on the `<operational>` datastore resource, described later in this document.

RESTCONF clients MAY also test if a server supports the NMDA using either the HEAD or GET methods on `'ietf-yang-library:yang-library'` resource, under either `{+restconf}/data` or `<operational>`, though only the latter resource SHOULD be used so that the client doesn't need to have any ongoing need to use the `{+restconf}/data` resource.

RESTCONF clients MAY also test if a server supports the NMDA by checking the revision number for the `"ietf-yang-library"` module listed under `'ietf-yang-library:modules-state'`, under either `{+restconf}/data` or `<operational>`. This approach might be preferred by some existing clients, but new clients should avoid using the deprecated `'modules-state'` resource.

5. The `{+restconf}/ds/<datastore>` Resource

Knowing which datastores a server supports, from querying the `ietf-yang-library` module, a RESTCONF client interacts with specific datastores using the resource path template:

```
{+restconf}/ds/<datastore>
```

Where `<datastore>` is encoded as an 'identity'. For instance:

```
{+restconf}/ds/ietf-datastores:running  
{+restconf}/ds/ietf-datastores:intended  
{+restconf}/ds/ietf-datastores:operational  
{+restconf}/ds/example-ds-ephemeral:ds-ephemeral
```

6. Protocol Operations

All existing protocol operations defined in [RFC8040] for the `{+restconf}/data` resource are available for all of the new datastore resources with the following exceptions:

- o Dynamic datastores are excluded, as each dynamic datastore definition needs to be reviewed for what protocol operations it supports.
- o Some datastores are read-only by nature (e.g., `<intended>`), and hence any attempt to modify these datastores will fail.
- o RFC 8040, Section 3.5.4, Paragraph 3 does not apply when interacting with `<operational>`.
- o New boolean query parameter 'with-origin' (default: false) is defined to request the 'origin' attributes when querying `<operational>`.

7. Security Considerations

TBD

8. IANA Considerations

TBD

9. Normative References

- [I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore
Architecture", draft-ietf-netmod-revised-datastores-02
(work in progress), May 2017.
- [I-D.nmdsdt-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module
Library", draft-nmdsdt-netconf-rfc7895bis-00 (work in
progress), May 2017.
- [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>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF
Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017,
<<http://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

EMail: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

EMail: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks

EMail: phil@juniper.net

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Rob Wilton
Cisco Systems

EMail: rwilton@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 10, 2017

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
May 9, 2017

Guidelines for YANG Module Authors (NMDA)
draft-dsdt-nmda-guidelines-01

Abstract

The "Network Management Datastore Architecture" (NMDA) adds the ability to inspect the current operational values for configuration, allowing clients to use identical paths for retrieving the configured values and the operational values. This change will simplify models and help modelers, but will create a period of transition as NMDA becomes a standard and is widely implemented. During that interim, the guidelines given in this document should help modelers find an optimal path forward.

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 November 10, 2017.

Copyright Notice

Copyright (c) 2017 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
1.1. Keywords	2
1.2. Terminology	2
1.3. Executive Summary	3
1.4. Background	3
1.5. Network Management Datastores Architecture	5
2. Guidelines for YANG Modelers	5
3. IANA Considerations	8
4. Security Considerations	8
5. Informative References	8
Authors' Addresses	9

1. Introduction

This document provides advice and guidelines to help modelers plan for the emerging "Network Management Datastore Architecture" (NMDA) [I-D.ietf-netmod-revised-datastores]. This architecture provides an architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [RFC8040] and the YANG [RFC7950] data modeling language. Datastores are a fundamental concept binding network management data models to network management protocols, enabling data models to be written in a network management protocol agnostic way.

1.1. Keywords

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.2. Terminology

This document uses the terminology defined by the NMDA [I-D.ietf-netmod-revised-datastores].

1.3. Executive Summary

The Network Management Datastore Architecture (NMDA) addresses the so called "OpState problem" that has been the subject of much discussion in the IETF. NMDA is still in development and there will be a transition period before NMDA solutions are universally available.

These guidelines are aimed to enable the creation of models that can take advantage of the NMDA, while pragmatically allowing those models to be used with the existing network configuration protocol implementations.

It is the strong recommendation that models SHOULD move as quickly as possible to the NMDA. The specific approach to be taken for models being developed now and during the NMDA transition period should be based on both the expected usage and the maturity of the data model.

1. New models and models that are not concerned with the operational state of configuration information SHOULD immediately be structured to be NMDA-compatible.
2. Models that require immediate support for "in use" and "system created" information SHOULD be structured for NMDA. A non-NMDA version of these models SHOULD also be published, using either an existing model or a model created either by hand or with suitable tools that support current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative appendix. The use of the non-NMDA model will allow temporary bridging of the time period until NMDA implementations are available.

Additional details on these guidelines can be found below, notably in Section 2.

1.4. Background

NETCONF ([RFC6241]) was developed with a focus on configuration data, and has unfortunate gaps in its treatment of operational data. The <get-config> operation can return configuration data (defined as nodes with "config true") stored in <running>. This data is typically the only data created by CLI users and NETCONF clients. The <get> operation is defined as returning all the data on the device, including the contents of <running>, as well as any operational state data. While the NETCONF design envisioned models merging "config false" nodes with the contents of running, there are two issues involved.

First, the desire of clients to see the true operational ("in use") value of configuration data resulted in the need for data models to have two distinct leafs, one to show the configured value and the other to show the operational value. An example would be the speed of an interface, where the configured value may not be the value that is currently used.

Second, devices often have "system created" resources that exist as operational data even when there is no corresponding configuration data. An example would be built-in networking interfaces that always appear in operational data.

A similar situation to the second issue discussed above exists while the device is processing configuration data changes. When configuration data is deleted, the operational data will continue to exist during the time period in which the device is releasing resources associated with the data. An example would be deleting a BGP peer, where the peer continues to exist in operational data until the connection is closed and any other resources are released.

To address these issues without requiring a protocol modification, two distinct strategies have been adopted in YANG model design:

The first strategy makes two distinct top-level containers, one for configuration and one for state. These are sometimes referred to as `"/foo"` and `"/foo-state"`. An example would be the interface model defined in [RFC7223]. These models require two completely distinct set of nodes, with repetition of both the interior containers, lists, and key nodes, but also repetition of many other nodes to allow visibility of the operational values of configured nodes. This leads to over-use of YANG groupings in ways that affect the readability of the models, as well as creating opportunities to incorrectly mirror the model's hierarchies. Also this "stitching together" of data from the two trees is merely a convention, not a formal relationship.

The second strategy uses two sibling containers, named `"config"` and `"state"`, placed deeper within the model node hierarchy. The `"config"` container holds the configured values, while the `"state"` container holds the operational values. The duplication of interior nodes in the hierarchies is removed, but the duplication of leafs representing configuration remains. Groupings can be used to avoid the repetition of the leafs in the YANG file, but at the expense of readability. In addition, this strategy does not handle the existence of operational data for which there is no configuration data, such as the system-created data and deleted peers scenarios discussed above.

1.5. Network Management Datastores Architecture

The Network Management Datastores Architecture (NMDA) addresses the problems mentioned above by creating an architectural framework which includes a distinct datastore for operational data, called `<operational>`. This datastore is defined as containing both config true and config false nodes, with the formal understanding that the "in use" values are returned for the config true nodes. This allows modelers to use a single hierarchy for all configuration and operational data, which both improves readability and reduces the possibility of modeling inconsistencies that might impact programmatic access.

In addition, another datastore named `<intended>` is defined to provide a complete view of the configuration data, even in the presence of device-specific features that expand or remove configuration data. While such mechanisms are currently non-standard, the NMDA recognizes they exist and need to be handled appropriately. In the future, such mechanisms may become standardized.

The NMDA allows the deprecation of NETCONF's `<get>` operation, removing the source of these issues. The new operations `<get-data>` and `<edit-data>` will support a parameter indicating the target datastore. Similar changes are planned for RESTCONF ([RFC8040]).

2. Guidelines for YANG Modelers

The following guidelines are meant to help modelers develop YANG models that will maximize the utility of the model with both current implementations and NMDA-capable implementations. Any questions regarding these guidelines can be sent to yang-doctors@ietf.org.

The direction taken should be based on both the maturity of the data model, along with the number of concrete implementations of the model. The intent is not to destabilize the IETF modeling community, but to create models that can take advantage of the NMDA, while pragmatically allowing those models to be used with the existing network configuration protocol implementations.

It is the strong recommendation that models SHOULD move as quickly as possible to the NMDA. This is key to the future of these models. The NETMOD WG will rework existing models to this architecture. Given the permanence and gravity of work published by the IETF, creating future-proof data models is vital.

The two current strategies ("`/foo-state`" and "`config/state`" containers) mix data retrieval details into the data model, complicating the models and impairing their readability. Rather than

maintain these details inside the data model, models can be post-processed to add this derivative information, either manually or using tools.

Tools can automatically produce the required derived modules. The suggested approach is to produce a "state" version of the module with a distinct namespace, rather than using the "/foo-state" top-level container. Since the contents are identical, constraints in the data model such as "must" statements should not need to change. Only the model name, namespace, and prefix should need to change. This simplifies the tooling needed to generate the derived model, as well as reducing changes needed in client applications when transitioning to the NMDA model.

These derived models use distinct module names and namespaces, allowing servers to announce their support for the base or derived models.

Consider the following trivial model:

```
module example-thermostat {
  namespace "tag:ietf:example:thermostat";
  prefix "thermo";

  container thermostat {
    leaf high-temperature {
      description "High temperature threshold";
      type int;
    }
    leaf low-temperature {
      description "Low temperature threshold";
      type int;
    }
    leaf current-temperature {
      description "Current temperature reading";
      type int;
      config false;
    }
  }
}
```

In the derived model, the contents mirror the NMDA data model, but are marked as "config false", and the module name and namespace values have a "-state" suffix:

```
module example-thermostat-state {
  namespace "tag:ietf:example:thermostat-state";
  prefix "thermo-state";

  container thermostat {
    config false;
    leaf high-temperature {
      description "High temperature threshold";
      type int;
    }
    leaf low-temperature {
      description "Low temperature threshold";
      type int;
    }
    leaf current-temperature {
      description "Current temperature reading";
      type int;
    }
  }
}
```

By adopting a tools-based solution for supporting models that are currently under development, models can be quickly restructured to be NMDA-compatible while giving continuity to their community of developers. When NMDA-capable implementations become available, the base data models can be used directly.

Modelers and reviewers can view the simple data model, published in the body of document. Tools can generate any required derived models, and those models can be published in a non-normative appendix to allow interoperability.

It is critical to consider the following guidelines, understanding that our goal is to make models that will see continued use in the long term, balancing short term needs against a desire for consistent, usable models in the future:

(a) New models and models that are not concerned with the operational state of configuration information SHOULD immediately be structured to be NMDA-compatible.

(b) Models that require immediate support for "in use" and "system created" information SHOULD be structured for NMDA. A non-NMDA version of these models SHOULD exist, either an existing model or a model created either by hand or with suitable tools that mirror the current modeling strategies. Both the NMDA and the non-NMDA modules SHOULD be published in the same document, with NMDA modules in the document main body and the non-NMDA modules in a non-normative

appendix. The use of the non-NMDA model will allow temporary bridging of the time period until NMDA implementations are available.

(c) For published models, the model should be republished with an NMDA-compatible structure, deprecating non-NMDA constructs. For example, the "ietf-interfaces" model in [RFC7223] will be restructured as an NMDA-compatible model. The "/interfaces-state" hierarchy will be marked "status deprecated". Models that mark their "/foo-state" hierarchy with "status deprecated" will allow NMDA-capable implementations to avoid the cost of duplicating the state nodes, while enabling non-NMDA-capable implementations to utilize them for access to the operational values.

(d) For models that augment models which have not been structured with the NMDA, the modeler will have to consider the structure of the base model and the guidelines listed above. Where possible, such models should move to new revisions of the base model that are NMDA-compatible. When that is not possible, augmenting "state" containers SHOULD be avoided, with the expectation that the base model will be re-released with the state containers marked as deprecated. It is RECOMMENDED to augment only the "/foo" hierarchy of the base model. Where this recommendation cannot be followed, then any new "state" elements SHOULD be included in their own module.

3. IANA Considerations

This document has no actions for IANA.

4. Security Considerations

This document has no security considerations.

5. Informative References

[I-D.ietf-netmod-revised-datastores]

Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture", draft-ietf-netmod-revised-datastores-01 (work in progress), March 2017.

[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>>.

- [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>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks

Email: phil@juniper.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Rob Wilton
Cisco Systems

Email: rwilton@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 5, 2018

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
July 4, 2017

NETCONF Model for NMDA
draft-dsdt-nmda-netconf-00

Abstract

The "Network Management Datastore Architecture" (NMDA) improves on NETCONF by adding new features to give more accurate handling of configuration and operational data. These include ability to inspect the current operational values of configuration data, allowing clients to use identical paths for retrieving the configured values and the operational values. These new features require additional operations in network management applications such as NETCONF. This draft details those new operations.

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 January 5, 2018.

Copyright Notice

Copyright (c) 2017 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
1.1. Keywords	3
1.2. Terminology	3
2. The NMDA Model for NETCONF	3
2.1. Operations	3
2.1.1. The <get-data> Operation	3
2.1.2. The <edit-data> Operation	4
2.2. Augmentations to the Base NETCONF Model	4
2.3. RPCs and Actions	4
3. YANG Model	5
4. IANA Considerations	9
5. Security Considerations	9
6. Informative References	9
Appendix A. Examples	9
Authors' Addresses	9

1. Introduction

This document provides a YANG model that adds NETCONF ([RFC6241]) support for the emerging "Network Management Datastore Architecture" (NMDA) [I-D.ietf-netmod-revised-datastores]. NMDA defines a framework for datastores, a fundamental concept binding network management data models to network management protocols, enabling data models to be written in a network management protocol agnostic way. NETCONF operations currently refer to the datastores defined in the original model, so new operations are required to allow references to the new datastores.

Operations like <copy-config>, <lock> and <unlock> are augmented to allow them to target additional datastores.

In addition the original <get> operation is deprecated, since the information it returns is no longer needed. <get>'s deficiencies were a major motivation for the NMDA.

1.1. Keywords

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here..

1.2. Terminology

This document uses the terminology defined by the NMDA [I-D.ietf-netmod-revised-datastores].

2. The NMDA Model for NETCONF

This section describes the changes needed for NMDA support. These changes are contained in a new YANG ([RFC7950]) model "ietf-netconf-datastores".

These changes include the use of source and target parameters based on the "datastore" identity defined in the "ietf-datastores" from [I-D.ietf-netmod-revised-datastores]. The use of identities allows future expansion in a way that the choice-based strategy from the original operations (e.g. <get-config>, <edit-config>) do not.

2.1. Operations

Support for the NMDA includes two new operations defined in this document.

2.1.1. The <get-data> Operation

The <get-data> operation retrieves data from a specific NMDA datastore. This operation is similar to NETCONF's "get-config" operation, but adds flexibility in naming the target datastore.

The "source" parameter indicates the datastore which is the source of the data to be retrieved. This is a datastore identity.

The "get-data" operation mirrors the "filter" parameter of the "get-config" operation, but it is modified to use "type anydata" for configuration content, rather than the "get-config"'s use of "type anyxml".

The "get-data" operation also supports the "with-defaults" parameter as defined in [RFC6243]. The supported values follow the constraints given by the "with-defaults" capability.

2.1.1.1. Origin Attribute

The "get-data" operation adds a new boolean parameter named "origin", which requests that the server return the "origin" information as detailed in the NMDA. This parameter is only valid for <operational> and any datastores with identities derived from the "operational" identity.

Data from <operational> can come from multiple sources. The server should return the most accurate value for the "origin" attribute as possible, indicating the source of the operational value.

When encoding the origin attribute for a hierarchy of returned nodes, the origin attribute may be omitted when the value matches that of the parent node.

2.1.2. The <edit-data> Operation

The <edit-data> operation changes the contents of a specific datastore, similar to the <edit-config> operation, but with additional flexibility in naming the target datastore.

The "target" parameter is a datastore identity that indicates the desired target datastore where changes should be made.

The "edit-content" parameter from "edit-config" it is modified to allow use "type anydata" for configuration content, rather than the "edit-config"'s use of "type anyxml".

The "default-operation" parameter mirrors the parameter of the "edit-config" operation.

2.2. Augmentations to the Base NETCONF Model

Several of the operations defined in the base NETCONF data model (ietf-netconf@2011-06-01.yang) will continue to be used under the NMDA. The <lock>, <unlock>, and <validate> operations are augmented with a new "datastore" leaf can indicate a desired NMDA datastore.

Only writable datastores can be locked.

2.3. RPCs and Actions

RPC operations and actions can be defined in YANG modules. The evaluation context for constraints and references in operation and actions is <operational>.

3. YANG Model

```
<CODE BEGINS> file "ietf-netconf-datastores@2017-06-30.yang"

module ietf-netconf-datastores {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-datastores";
  prefix ncds;

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-datastores {
    prefix ds;
  }
  import ietf-netconf {
    prefix nc;
  }
  import ietf-netconf-with-defaults {
    prefix ncwd;
  }

  organization
    "IETF NETCONF Working Group";
  contact
    "WG Web:    <https://datatracker.ietf.org/wg/netconf/>

    WG List:    <mailto:netconf@ietf.org>

    Author:     Martin Bjorklund
                <mailto:mbj@tail-f.com>

    Author:     Juergen Schoenwaelder
                <mailto:j.schoenwaelder@jacobs-university.de>

    Author:     Phil Shafer
                <mailto:phil@juniper.net>

    Author:     Kent Watsen
                <mailto:kwatsen@juniper.net>

    Author:     Rob Wilton
                <rwilton@cisco.com>";
  description
    "This YANG module defines a set of NETCONF operations for the
    Network Management Datastore Architecture (NMDA).

    Copyright (c) 2017 IETF Trust and the persons identified as
```

authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<http://www.rfc-editor.org/info/rfcxxxx>); see the RFC itself for full legal notices."

```
revision 2017-06-30 {
  description
    "Initial revision.";
  reference "RFC XXXX: NETCONF Support for NMDA";
}

typedef datastore {
  type identityref {
    base ds:datastore;
  }
  description
    "An NMDA datastore.";
  reference "RFC XXXX: Network Management Datastore Architecture";
}

rpc get-data {
  description
    "Get data from an NMDA datastore";
  input {
    leaf source {
      type ncds:datastore;
      description
        "Datastore from which to retrieve data.";
    }
    anydata filter {
      description
        "Subtree or XPath filter to use.";
      nc:get-filter-element-attributes;
    }
  }
  output {
    anydata data {
      description
        "Copy of the source datastore subset which matched
        the filter criteria (if any). An empty data
```

```

        container indicates that the request did not
        produce any results.";
    }
}
}
rpc edit-data {
  description
    "Edit data in an NMDA datastore.";
  input {
    leaf target {
      type ncds:datastore;
      description
        "Datastore which data affects.";
    }
    leaf default-operation {
      type enumeration {
        enum "merge" {
          description
            "The default operation is merge.";
        }
        enum "replace" {
          description
            "The default operation is replace.";
        }
        enum "none" {
          description
            "There is no default operation.";
        }
      }
      default "merge";
      description
        "The default operation to use.";
    }
  }
  uses ncwd:with-defaults-parameters;
  choice edit-content {
    mandatory true;
    description
      "The content for the edit operation.";

    anydata config {
      description
        "Inline Config content.";
    }
    leaf url {
      if-feature nc:url;
      type inet:uri;
      description
        "URL based config content.";
    }
  }
}

```

```

    }
  }
}

/*
 * Augment the lock and unlock operations with a
 * "datastore" parameter.
 */

augment "/nc:lock/nc:input/nc:target/nc:config-target" {
  description
    "Add NMDA Datastore as target.";
  leaf datastore {
    type ncds:datastore;
    description
      "Datastore to lock.";
  }
}

augment "/nc:unlock/nc:input/nc:target/nc:config-target" {
  description
    "Add NMDA Datastore as target.";
  leaf datastore {
    type ncds:datastore;
    description
      "Datastore to unlock.";
  }
}

/*
 * Augment the validate operation with a
 * "datastore" parameter.
 */

augment "/nc:validate/nc:input/nc:source/nc:config-source" {
  description
    "Add NMDA Datastore as source.";
  leaf datastore {
    type ncds:datastore;
    description
      "Datastore to validate.";
  }
}
}

<CODE ENDS>

```

4. IANA Considerations

This document has no actions for IANA.

5. Security Considerations

This document has no security considerations.

6. Informative References

[I-D.ietf-netmod-revised-datastores]

Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore
Architecture", draft-ietf-netmod-revised-datastores-03
(work in progress), July 2017.

[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>>.

[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>>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.

Appendix A. Examples

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks

Email: phil@juniper.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Robert Wilton
Cisco Systems

Email: rwilton@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 15, 2017

K. Watsen
Juniper Networks
June 13, 2017

Keystore Model
draft-ietf-netconf-keystore-02

Abstract

This document defines a YANG data module for a system-level keystore mechanism, that might be used to hold onto private keys and certificates that are trusted by the system advertising support for this module.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- o "VVVV" --> the assigned RFC value for this draft

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- o "2017-06-13" --> the publication date of this draft

The following Appendix section is to be removed prior to publication:

- o Appendix A. Change Log

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 December 15, 2017.

Copyright Notice

Copyright (c) 2017 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	3
1.1. Requirements Language	3
1.2. Tree Diagram Notation	3
2. The Keystore Model	4
2.1. Overview	4
2.2. Example Usage	5
2.3. YANG Module	10
3. Design Considerations	21
4. Security Considerations	22
5. IANA Considerations	23
5.1. The IETF XML Registry	23
5.2. The YANG Module Names Registry	23
6. Acknowledgements	24
7. References	24
7.1. Normative References	24
7.2. Informative References	25
Appendix A. Change Log	26
A.1. server-model-09 to 00	26
A.2. keychain-00 to keystore-00	26
A.3. 00 to 01	26
A.4. 01 to 02	26
Author's Address	26

1. Introduction

This document defines a YANG [RFC6020] data module for a system-level keystore mechanism, which can be used to hold onto private keys and certificates that are trusted by the system advertising support for this module.

This module provides a centralized location for security sensitive data, so that the data can be then referenced by other modules. There are two types of data that are maintained by this module:

- o Private keys, and any associated public certificates.
- o Sets of trusted certificates.

This document extends special consideration for systems that have Trusted Protection Modules (TPMs). These systems are unique in that the TPM must be directed to generate new private keys (it is not possible to load a private key into a TPM) and it is not possible to backup/restore the TPM's private keys as configuration.

It is not required that a system has an operating system level keystore utility to implement this module.

1.1. Requirements Language

The keywords "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].

1.2. Tree Diagram Notation

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. The Keystore Model

The keystore module defined in this section provides a configurable object having the following characteristics:

- o A semi-configurable list of private keys, each with one or more associated certificates. Private keys **MUST** be either preinstalled (e.g., a key associated to an IDevID [Std-802.1AR-2009] certificate), be generated by request, or be loaded by request. Each private key is **MAY** have associated certificates, either preinstalled or configured after creation.
- o A configurable list of lists of trust anchor certificates. This enables the server to have use-case specific trust anchors. For instance, one list of trust anchors might be used to authenticate management connections (e.g., client certificate-based authentication for NETCONF or RESTCONF connections), and a different list of trust anchors might be used for when connecting to a specific Internet-based service (e.g., a zero touch bootstrap server).
- o An RPC to generate a certificate signing request for an existing private key, a passed subject, and an optional attributes. The signed certificate returned from an external certificate authority (CA) can be later set using a standard configuration change request (e.g., <edit-config>).
- o An RPC to request the server to generate a new private key using the specified algorithm and key length.
- o An RPC to request the server to load a new private key.

2.1. Overview

The keystore module has the following tree diagram. Please see Section 1.2 for information on how to interpret this diagram.

```

module: ietf-keystore
  +--rw keystore
    +--rw keys
      +--rw key* [name]
        +--rw name string
        +--rw algorithm-identifier identityref
        +--rw private-key union
        +--ro public-key binary
        +--rw certificates
          +--rw certificate* [name]
            +--rw name string
            +--rw value? binary
          +---x generate-certificate-signing-request
            +---w input
              +---w subject binary
              +---w attributes? binary
            +--ro output
              +--ro certificate-signing-request binary
          +---x generate-private-key
            +---w input
              +---w name string
              +---w algorithm identityref
      +--rw trusted-certificates* [name]
        +--rw name string
        +--rw description? string
        +--rw trusted-certificate* [name]
          +--rw name string
          +--rw certificate? binary
      +--rw trusted-host-keys* [name]
        +--rw name string
        +--rw description? string
        +--rw trusted-host-key* [name]
          +--rw name string
          +--rw host-key binary

notifications:
  +---n certificate-expiration
    +--ro certificate instance-identifier
    +--ro expiration-date yang:date-and-time

```

2.2. Example Usage

The following example illustrates what a fully configured keystore object might look like. The private-key shown below is consistent with the generate-private-key and generate-certificate-signing-request examples above. This example also assumes that the resulting CA-signed certificate has been configured back onto the server.

Lastly, this example shows that three lists of trusted certificates having been configured.

```
<keystore xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">

  <!-- private keys and associated certificates -->
  <keys>
    <key>
      <name>ex-rsa-key</name>
      <algorithm-identifier>rsa1024</algorithm-identifier>
      <private-key>Base64-encoded RSA Private Key</private-key>
      <public-key>Base64-encoded RSA Public Key</public-key>
      <certificates>
        <certificate>
          <name>ex-rsa-cert</name>
          <value>Base64-encoded PKCS#7</value>
        </certificate>
      </certificates>
    </key>

    <key>
      <name>tls-ec-key</name>
      <algorithm-identifier>secp256r1</algorithm-identifier>
      <private-key>Base64-encoded EC Private Key</private-key>
      <public-key>Base64-encoded EC Public Key</public-key>
      <certificates>
        <certificate>
          <name>tls-ec-cert</name>
          <value>Base64-encoded PKCS#7</value>
        </certificate>
      </certificates>
    </key>

    <key>
      <name>tpm-protected-key</name>
      <algorithm-identifier>rsa2048</algorithm-identifier>
      <private-key>Base64-encoded RSA Private Key</private-key>
      <public-key>Base64-encoded RSA Public Key</public-key>
      <certificates>
        <certificate>
          <name>builtin-idevid-cert</name>
          <value>Base64-encoded PKCS#7</value>
        </certificate>
        <certificate>
          <name>my-ldevid-cert</name>
          <value>Base64-encoded PKCS#7</value>
        </certificate>
      </certificates>
    </key>
  </keys>
</keystore>
```

```
</key>
</keys>

<!-- trusted netconf/restconf client certificates -->
<trusted-certificates>
  <name>explicitly-trusted-client-certs</name>
  <description>
    Specific client authentication certificates for explicitly
    trusted clients. These are needed for client certificates
    that are not signed by a trusted CA.
  </description>
  <trusted-certificate>
    <name>George Jetson</name>
    <certificate>Base64-encoded X.509v3</certificate>
  </trusted-certificate>
</trusted-certificates>

<trusted-certificates>
  <name>explicitly-trusted-server-certs</name>
  <description>
    Specific server authentication certificates for explicitly
    trusted servers. These are needed for server certificates
    that are not signed by a trusted CA.
  </description>
  <trusted-certificate>
    <name>Fred Flintstone</name>
    <certificate>Base64-encoded X.509v3</certificate>
  </trusted-certificate>
</trusted-certificates>

<!-- trust anchors (CA certs) for authenticating clients -->
<trusted-certificates>
  <name>deployment-specific-ca-certs</name>
  <description>
    Trust anchors (i.e. CA certs) that are used to authenticate
    client connections. Clients are authenticated if their
    certificate has a chain of trust to one of these configured
    CA certificates.
  </description>
  <trusted-certificate>
    <name>ca.example.com</name>
    <certificate>Base64-encoded X.509v3</certificate>
  </trusted-certificate>
</trusted-certificates>

<!-- trust anchors for random HTTPS servers on Internet -->
<trusted-certificates>
  <name>common-ca-certs</name>
```



```

    <description>
      Trusted certificates to authenticate common HTTPS servers.
      These certificates are similar to those that might be
      shipped with a web browser.
    </description>
    <trusted-certificate>
      <name>ex-certificate-authority</name>
      <certificate>Base64-encoded X.509v3</certificate>
    </trusted-certificate>
  </trusted-certificates>

  <!-- trusted SSH host keys -->
  <trusted-host-keys>
    <name>explicitly-trusted-ssh-host-keys</name>
    <description>
      Trusted SSH host keys used to authenticate SSH servers.
      These host keys would be analogous to those stored in
      a known_hosts file in OpenSSH.
    </description>
    <trusted-host-key>
      <name>corp-fw1</name>
      <host-key>Base64-encoded OneAsymmetricKey</host-key>
    </trusted-host-key>
  </trusted-host-keys>

</keystore>

```

The following example illustrates the "generate-certificate-signing-request" action in use with the NETCONF protocol.

REQUEST

```

<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <keystore
      xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
      <keys>
        <key>
          <name>ex-key-sect571r1</name>
          <generate-certificate-signing-request>
            <subject>
              cztvaWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvO2R
              manZvO3NkZmJpdmhZGZpbHVidjtvZ2lkZmhidmllbHNlmo
              Z2aXNiZGZpYmhZG87ZmJvO3NkZ25iO29pLmR6Zgo=
            </subject>
            <attributes>

```

```

        bwtakWRoc2RmZ2tqaHNkZmdramRzZnZzZGtmam5idnNvut4
        arnZvO3NkZmJpdmhZGZpbHvidjtvC2lkZmhidml1bHNkYm
        Z2aXNiZGZpYmhZG87ZmJvO3NkZ25iO29pLmC6Rhp=
    </attributes>
  </generate-certificate-signing-request>
</key>
</keys>
</keystore>
</action>
</rpc>

```

RESPONSE

```

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <certificate-signing-request
    xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
    LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUNrekNDQWZ5Z
    0F3SUJBZ01kQUprT2t3bGpNK2pjTUEwR0NTcUdTSWIZRFFFQkJRvU
    FNRFF4Q3pBSk1JNk1YkQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
    GdGNHeGxNUk13RVFZRFZRUURFd3BEVWt3ZlNYTnpkV1Z5TUI0WApe
    diR1V4RXpBUk1JNk1YkQkFZVEFsVlRNUkF3RGdZRFZRUUtFd2RsZUd
    KS29aSW2Y04KQVFFQk1RQURnWTBBTUlHSkFvR0JBTXVvZmFPNEV3
    El1QWMrQ1RStkNmc0d6cEw1Um5ydXZsOFRlcUJtdGZQY3N0Zk1KT1
    FaNzlnNlNWVldsMldzaHE1bUViCk1JNk1YkQkFZVEFsVlRNUkF3RGd
    bXBDT2YkQWdNQkFBR2pnyXZ3Z2Frd0hRWURWUjBPQkJZRUZKY1o2W
    URiR01PNDB4ajlPb3JtREdsRUNCVTfNR1FHQTFVZApJd1JkTUZ1QU
    ZKY1o2WURiR01PNDB4ajlPb3JtREdsRUNCVTfVVGlrTm1pBME1Rc3d
    mMKTUE0R0ExVWREd0VCL3dRRUF3SUNCREFTQmdOVkhSTUJBZjhFQ0
    RBR0FRSC9BZ0VBTUEwR0NTcUdTSWIZRFFFQgpCUVVBQTRHkFMMmx
    rWmFGNWcyAGR6MVNhZnZPbnBneHA4eG00SHRhbStadHpLazFls3Bx
    TXp4YXJCbFpDSH1LCk1VbC9GVzRtV1RQS1VDeEtFTE40NEY2Zmk2d
    c4d0tSSElkYW1WL0pGTmlQS0VXSTF4K1IlaDZmazcrQzQ1QXglRWV
    SWHgzZjdVM2xZTgotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
  </certificate-signing-request>
</rpc-reply>

```

The following example illustrates the "generate-private-key" action in use with the RESTCONF protocol and JSON encoding.

REQUEST

['\`' line wrapping added for formatting only]

```
POST https://example.com/restconf/data/ietf-keystore:keystore/\
keys/generate-private-key HTTP/1.1
HOST: example.com
Content-Type: application/yang.operation+json
```

```
{
  "ietf-keystore:input" : {
    "name" : "ex-key-sect571r1",
    "algorithm" : "sect571r1"
  }
}
```

RESPONSE

```
HTTP/1.1 204 No Content
Date: Mon, 31 Oct 2015 11:01:00 GMT
Server: example-server
```

The following example illustrates a "certificate-expiration" notification in XML.

['\`' line wrapping added for formatting only]

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2016-07-08T00:01:00Z</eventTime>
  <certificate-expiration
    xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
    <certificate>/ks:keystore/ks:private-keys/ks:private-key\
/ks:certificate-chains/ks:certificate-chain/ks:certificate[3]\
    </certificate>
    <expiration-date>2016-08-08T14:18:53-05:00</expiration-date>
  </certificate-expiration>
</notification>
```

2.3. YANG Module

This YANG module makes extensive use of data types defined in [RFC5280] and [RFC5958].

```
<CODE BEGINS> file "ietf-keystore@2017-06-13.yang"

module ietf-keystore {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-keystore";
  prefix "ks";

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 6536: Network Configuration Protocol (NETCONF) Access
      Control Model";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>

    Author:     Kent Watsen
                <mailto:kwatsen@juniper.net>";

  description
    "This module defines a keystore to centralize management
    of security credentials.

    Copyright (c) 2014 IETF Trust and the persons identified
    as authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with
    or without modification, is permitted pursuant to, and
    subject to the license terms contained in, the Simplified
    BSD License set forth in Section 4.c of the IETF Trust's
    Legal Provisions Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC VVVV; see
    the RFC itself for full legal notices.";
```

```
revision "2017-06-13" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
      Models";
}

// Identities

identity key-algorithm {
  description
    "Base identity from which all key-algorithms are derived.";
}

identity rsa1024 {
  base key-algorithm;
  description
    "The RSA algorithm using a 1024-bit key.";
  reference
    "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
      RSA Cryptography Specifications Version 2.1.";
}

identity rsa2048 {
  base key-algorithm;
  description
    "The RSA algorithm using a 2048-bit key.";
  reference
    "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
      RSA Cryptography Specifications Version 2.1.";
}

identity rsa3072 {
  base key-algorithm;
  description
    "The RSA algorithm using a 3072-bit key.";
  reference
    "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
      RSA Cryptography Specifications Version 2.1.";
}

identity rsa4096 {
  base key-algorithm;
  description
    "The RSA algorithm using a 4096-bit key.";
  reference
    "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
```

```
        RSA Cryptography Specifications Version 2.1.";
    }

    identity rsa7680 {
        base key-algorithm;
        description
            "The RSA algorithm using a 7680-bit key.";
        reference
            "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
              RSA Cryptography Specifications Version 2.1.";
    }

    identity rsa15360 {
        base key-algorithm;
        description
            "The RSA algorithm using a 15360-bit key.";
        reference
            "RFC3447: Public-Key Cryptography Standards (PKCS) #1:
              RSA Cryptography Specifications Version 2.1.";
    }

    identity secp192r1 {
        base key-algorithm;
        description
            "The secp192r1 algorithm.";
        reference
            "RFC5480:
              Elliptic Curve Cryptography Subject Public Key Information.";
    }

    identity secp256r1 {
        base key-algorithm;
        description
            "The secp256r1 algorithm.";
        reference
            "RFC5480:
              Elliptic Curve Cryptography Subject Public Key Information.";
    }

    identity secp384r1 {
        base key-algorithm;
        description
            "The secp384r1 algorithm.";
        reference
            "RFC5480:
              Elliptic Curve Cryptography Subject Public Key Information.";
    }
}
```

```
identity secp521r1 {
  base key-algorithm;
  description
    "The secp521r1 algorithm.";
  reference
    "RFC5480:
      Elliptic Curve Cryptography Subject Public Key Information.";
}

// data model

container keystore {
  nacm:default-deny-write;
  description
    "The keystore contains both active material (e.g., private keys
    and passwords) and passive material (e.g., trust anchors).

    The active material can be used to support either a server (e.g.,
    a TLS/SSH server's private) or a client (a private key used for
    TLS/SSH client-certificate based authentication, or a password
    used for SSH/HTTP-client authentication).

    The passive material can be used to support either a server
    (e.g., client certificates to trust) or clients (e.g., server
    certificates to trust).";

  container keys {
    description
      "A list of keys maintained by the keystore.";
    list key {
      key name;
      description
        "A key maintained by the keystore.";
      leaf name {
        type string;
        description
          "An arbitrary name for the key.";
      }
      leaf algorithm-identifier {
        type identityref {
          base "key-algorithm";
        }
        mandatory true;
        description
          "Identifies which algorithm is to be used to generate the
          key.";
        // no 'params' like in RFC 5912? - none are set for
        // algs we care about, but what about the future?
      }
    }
  }
}
```

```
}
leaf private-key {
  nacm:default-deny-all;
  type union {
    type binary;
    type enumeration {
      enum "INACCESSIBLE" {
        description
          "The private key is inaccessible due to being protected
          by a cryptographic hardware module (e.g., a TPM).";
      }
    }
  }
}
mandatory true;
description
  "A binary string that contains the value of the private
  key. The interpretation of the content is defined in the
  registration of the key algorithm. For example, a DSA key
  is an INTEGER, an RSA key is represented as RSAPrivateKey
  as defined in [RFC3447], and an Elliptic Curve Cryptography
  (ECC) key is represented as ECPrivateKey as defined in
  [RFC5915]"; // text lifted from RFC5958
}

// no key usage (ref: RFC 5912, pg 101 -- too X.509 specific?)

leaf public-key {
  type binary;
  config false;
  mandatory true;
  description
    "A binary string that contains the value of the public
    key. The interpretation of the content is defined in the
    registration of the key algorithm. For example, a DSA key
    is an INTEGER, an RSA key is represented as RSAPublicKey
    as defined in [RFC3447], and an Elliptic Curve Cryptography
    (ECC) key is represented using the 'publicKey' described in
    [RFC5915]";
}
container certificates {
  description
    "Certificates associated with this private key. More
    than one certificate per key is enabled to support,
    for instance, a TPM-protected key that has associated
    both IDevID and LDevID certificates.";
  list certificate {
    key name;
    description
```



```
    "A certificate for this private key.";
  leaf name {
    type string;
    description
      "An arbitrary name for the certificate. The name
       must be a unique across all keys, not just within
       this key.";
  }
  leaf value {
    type binary;
    description
      "An unsigned PKCS #7 SignedData structure, as specified
       by Section 9.1 in RFC 2315, containing just certificates
       (no content, signatures, or CRLs), encoded using ASN.1
       distinguished encoding rules (DER), as specified in
       ITU-T X.690.

       This structure contains, in order, the certificate
       itself and all intermediate certificates leading up
       to a trust anchor certificate. The certificate MAY
       optionally include the trust anchor certificate.";
    reference
      "RFC 2315:
       PKCS #7: Cryptographic Message Syntax Version 1.5.
       ITU-T X.690:
       Information technology - ASN.1 encoding rules:
       Specification of Basic Encoding Rules (BER),
       Canonical Encoding Rules (CER) and Distinguished
       Encoding Rules (DER).";
  }
}
}
}
action generate-certificate-signing-request {
  description
    "Generates a certificate signing request structure for
     the associated private key using the passed subject and
     attribute values. The specified assertions need to be
     appropriate for the certificate's use. For example,
     an entity certificate for a TLS server SHOULD have
     values that enable clients to satisfy RFC 6125
     processing.";
  input {
    leaf subject {
      type binary;
      mandatory true;
      description
        "The 'subject' field from the CertificationRequestInfo
         structure as specified by RFC 2986, Section 4.1 encoded
```

```
        using the ASN.1 distinguished encoding rules (DER), as
        specified in ITU-T X.690.";
    reference
        "RFC 2986:
        PKCS #10: Certification Request Syntax Specification
        Version 1.7.
        ITU-T X.690:
        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
}
leaf attributes {
    type binary;
    description
        "The 'attributes' field from the CertificationRequestInfo
        structure as specified by RFC 2986, Section 4.1 encoded
        using the ASN.1 distinguished encoding rules (DER), as
        specified in ITU-T X.690.";
    reference
        "RFC 2986:
        PKCS #10: Certification Request Syntax Specification
        Version 1.7.
        ITU-T X.690:
        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
}
}
output {
    leaf certificate-signing-request {
        type binary;
        mandatory true;
        description
            "A CertificationRequest structure as specified by RFC
            2986, Section 4.1 encoded using the ASN.1 distinguished
            encoding rules (DER), as specified in ITU-T X.690.";
        reference
            "RFC 2986:
            PKCS #10: Certification Request Syntax Specification
            Version 1.7.
            ITU-T X.690:
            Information technology - ASN.1 encoding rules:
            Specification of Basic Encoding Rules (BER),
            Canonical Encoding Rules (CER) and Distinguished
            Encoding Rules (DER).";
    }
}
```

```
    }
  }
} // end key

action generate-private-key {
  description
    "Requests the device to generate a private key using the
    specified key algorithm. This action is primarily to
    support cryptographic processors that MUST generate
    the private key themselves. The resulting key is
    considered operational state and hence initially only
    present in the <operational> datastore, as defined in
    [I-D.netmod-revised-datastores].";
  input {
    leaf name {
      type string;
      mandatory true;
      description
        "The name this private-key should have when listed
        in /keys/key. As such, the passed value MUST NOT
        match any existing 'name' value.";
    }
    leaf algorithm {
      type identityref {
        base "key-algorithm";
      }
      mandatory true;
      description
        "The algorithm to be used when generating the key.";
    }
  }
} // end generate-private-key
} // end keys

list trusted-certificates {
  key name;
  description
    "A list of trusted certificates. These certificates
    can be used by a server to authenticate clients, or by
    clients to authenticate servers. The certificates may
    be endpoint specific or for certificate authorities,
    to authenticate many clients at once. Each list of
    certificates SHOULD be specific to a purpose, as the
    list as a whole may be referenced by other modules.
    For instance, a NETCONF server model might point to
    a list of certificates to use when authenticating
    client certificates.";
```

```

    leaf name {
        type string;
        description
            "An arbitrary name for this list of trusted certificates.";
    }
    leaf description {
        type string;
        description
            "An arbitrary description for this list of trusted
            certificates.";
    }
    list trusted-certificate {
        key name;
        description
            "A trusted certificate for a specific use. Note, this
            'certificate' is a list in order to encode any
            associated intermediate certificates.";
        leaf name {
            type string;
            description
                "An arbitrary name for this trusted certificate. Must
                be unique across all lists of trusted certificates
                (not just this list) so that a leafref to it from
                another module can resolve to unique values.";
        }
        leaf certificate { // rename to 'data'?
            type binary;
            description
                "An X.509 v3 certificate structure as specified by RFC
                5280, Section 4 encoded using the ASN.1 distinguished
                encoding rules (DER), as specified in ITU-T X.690.";
            reference
                "RFC 5280:
                Internet X.509 Public Key Infrastructure Certificate
                and Certificate Revocation List (CRL) Profile.
                ITU-T X.690:
                Information technology - ASN.1 encoding rules:
                Specification of Basic Encoding Rules (BER),
                Canonical Encoding Rules (CER) and Distinguished
                Encoding Rules (DER).";
        }
    }
}

list trusted-host-keys {
    key name;
    description
        "A list of trusted host-keys. These host-keys can be used

```

by clients to authenticate SSH servers. The host-keys are endpoint specific. Each list of host-keys SHOULD be specific to a purpose, as the list as a whole may be referenced by other modules. For instance, a NETCONF client model might point to a list of host-keys to use when authenticating servers host-keys.";

```
leaf name {
  type string;
  description
    "An arbitrary name for this list of trusted SSH host keys.";
}
leaf description {
  type string;
  description
    "An arbitrary description for this list of trusted SSH host
    keys.";
}
list trusted-host-key {
  key name;
  description
    "A trusted host key.";
  leaf name {
    type string;
    description
      "An arbitrary name for this trusted host-key. Must be
      unique across all lists of trusted host-keys (not just
      this list) so that a leafref to it from another module
      can resolve to unique values.

      Note that, for when the SSH client is able to listen
      for call-home connections as well, there is no reference
      identifier (e.g., hostname, IP address, etc.) that it
      can use to uniquely identify the server with. The
      call-home draft recommends SSH servers use X.509v3
      certificates (RFC6187) when calling home.";
  }
}
leaf host-key { // rename to 'data'?
  type binary;
  mandatory true;
  description // is this the correct type?
    "An OneAsymmetricKey 'publicKey' structure as specified
    by RFC 5958, Section 2 encoded using the ASN.1
    distinguished encoding rules (DER), as specified
    in ITU-T X.690.";
  reference
    "RFC 5958:
      Asymmetric Key Packages
      ITU-T X.690:"
```

```

        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
    }
}
}

notification certificate-expiration {
  description
    "A notification indicating that a configured certificate is
    either about to expire or has already expired.  When to send
    notifications is an implementation specific decision, but
    it is RECOMMENDED that a notification be sent once a month
    for 3 months, then once a week for four weeks, and then once
    a day thereafter.";
  leaf certificate {
    type instance-identifier;
    mandatory true;
    description
      "Identifies which certificate is expiring or is expired.";
  }
  leaf expiration-date {
    type yang:date-and-time;
    mandatory true;
    description
      "Identifies the expiration date on the certificate.";
  }
}
}

```

<CODE ENDS>

3. Design Considerations

This document uses PKCS #10 [RFC2986] for the "generate-certificate-signing-request" action. The use of Certificate Request Message Format (CRMF) [RFC4211] was considered, but it was unclear if there was market demand for it, and so support for CRMF has been left out of this specification. If it is desired to support CRMF in the future, placing a "choice" statement in both the input and output statements, along with an "if-feature" statement on the CRMF option, would enable a backwards compatible solution.

This document puts a limit of the number of elliptical curves supported by default. This was done to match industry trends in IETF best practice (e.g., matching work being done in TLS 1.3). If additional algorithms are needed, they MAY be augmented in by another module, or added directly in a future version of this document.

For the trusted-certificates list, Trust Anchor Format [RFC5914] was evaluated and deemed inappropriate due to this document's need to also support pinning. That is, pinning a client-certificate to support NETCONF over TLS client authentication.

4. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/: The entire data tree defined by this module is sensitive to write operations. For instance, the addition or removal of keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. This being the case, the top-level node in this module is marked with the NACM value 'default-deny-write'.

/keystore/keys/key/private-key: When writing this node, implementations MUST ensure that the strength of the key being configured is not greater than the strength of the underlying secure transport connection over which it is communicated. Implementations SHOULD fail the write-request if ever the strength of the private key is greater than the strength of the underlying transport, and alert the client that the strength of the key may have been compromised. Additionally, when deleting this node, implementations SHOULD automatically (without explicit request) zeroize these keys in the most secure manner

available, so as to prevent the remnants of their persisted storage locations from being analyzed in any meaningful way.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

/keystore/keys/key/private-key: This node is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. The best reason for returning this node is to support backup/restore type workflows. This being the case, this node is marked with the NACM value 'default-deny-all'.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

generate-certificate-signing-request: For this RPC operation, it is RECOMMENDED that implementations assert channel binding [RFC5056], so as to ensure that the application layer that sent the request is the same as the device authenticated when the secure transport layer was established.

5. IANA Considerations

5.1. The IETF XML Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-keystore
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

This document registers one YANG module in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the the following registration is requested:

name: ietf-keystore
namespace: urn:ietf:params:xml:ns:yang:ietf-keystore
prefix: kc
reference: RFC VVVV

6. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, Balazs Kovacs, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Juergen Schoenwaelder; Phil Shafer, Sean Turner, and Bert Wijnen.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <<http://www.rfc-editor.org/info/rfc2986>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<http://www.rfc-editor.org/info/rfc5280>>.
- [RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <<http://www.rfc-editor.org/info/rfc5958>>.
- [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>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

7.2. Informative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4211] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <<http://www.rfc-editor.org/info/rfc4211>>.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <<http://www.rfc-editor.org/info/rfc5056>>.
- [RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <<http://www.rfc-editor.org/info/rfc5914>>.
- [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>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.
- [Std-802.1AR-2009] IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Removed key-usage parameter from generate-private-key action.
- o Now /private-keys/private-key/certificates/certificate/name must be globally unique (unique across all private keys).
- o Added top-level 'trusted-ssh-host-keys' and 'user-auth-credentials' to support SSH client modules.

A.2. keychain-00 to keystore-00

- o Renamed module from "keychain" to "keystore" (Issue #3)

A.3. 00 to 01

- o Replaced the 'certificate-chain' structures with PKCS#7 structures. (Issue #1)
- o Added 'private-key' as a configurable data node, and removed the 'generate-private-key' and 'load-private-key' actions. (Issue #2)
- o Moved 'user-auth-credentials' to the ietf-ssh-client module. (Issues #4 and #5)

A.4. 01 to 02

- o Added back 'generate-private-key' action.
- o Removed 'RESTRICTED' enum from the 'private-key' leaf type.
- o Fixed up a few description statements.

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

K. Watsen
Juniper Networks
G. Wu
Cisco Networks
J. Schoenwaelder
Jacobs University Bremen
July 3, 2017

NETCONF Client and Server Models
draft-ietf-netconf-netconf-client-server-04

Abstract

This document defines two YANG modules, one module to configure a NETCONF client and the other module to configure a NETCONF server. Both modules support both the SSH and TLS transport protocols, and support both standard NETCONF and NETCONF Call Home connections.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

This document contains references to other drafts in progress, both in the Normative References section, as well as in body text throughout. Please update the following references to reflect their final RFC assignments:

- o I-D.ietf-netconf-keystore
- o I-D.ietf-netconf-ssh-client-server
- o I-D.ietf-netconf-tls-client-server

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- o "XXXX" --> the assigned RFC value for this draft
- o "YYYY" --> the assigned RFC value for I-D.ietf-netconf-ssh-client-server
- o "ZZZZ" --> the assigned RFC value for I-D.ietf-netconf-tls-client-server

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- o "2017-07-03" --> the publication date of this draft

The following Appendix section is to be removed prior to publication:

- o Appendix A. Change Log

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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 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	4
1.1. Terminology	4
1.2. Tree Diagrams	4
2. The NETCONF Client Model	4
2.1. Tree Diagram	5
2.2. Example Usage	8
2.3. YANG Model	10
3. The NETCONF Server Model	20
3.1. Tree Diagram	20
3.2. Example Usage	23
3.3. YANG Model	26
4. Design Considerations	37
4.1. Support all NETCONF transports	38
4.2. Enable each transport to select which keys to use	38
4.3. Support authenticating NETCONF clients certificates	38
4.4. Support mapping authenticated NETCONF client certificates to usernames	38
4.5. Support both listening for connections and call home	38
4.6. For Call Home connections	39
4.6.1. Support more than one NETCONF client	39
4.6.2. Support NETCONF clients having more than one endpoint	39
4.6.3. Support a reconnection strategy	39
4.6.4. Support both persistent and periodic connections	39
4.6.5. Reconnection strategy for periodic connections	40
4.6.6. Keep-alives for persistent connections	40
4.6.7. Customizations for periodic connections	40
5. Security Considerations	40
6. IANA Considerations	41
6.1. The IETF XML Registry	41
6.2. The YANG Module Names Registry	42
7. Acknowledgements	42
8. References	42
8.1. Normative References	42
8.2. Informative References	43
Appendix A. Change Log	45
A.1. server-model-09 to 00	45
A.2. 00 to 01	45
A.3. 01 to 02	45
A.4. 02 to 03	45
A.5. 03 to 04	45
Authors' Addresses	46

1. Introduction

This document defines two YANG [RFC7950] modules, one module to configure a NETCONF client and the other module to configure a NETCONF server. Both modules support both the SSH and TLS transport protocols, and support both standard NETCONF and NETCONF Call Home connections.

NETCONF is defined by [RFC6241]. SSH is defined by [RFC4252], [RFC4253], and [RFC4254]. TLS is defined by [RFC5246]. NETCONF Call Home is defined by [RFC8071]).

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Tree Diagrams

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. The NETCONF Client Model

The NETCONF client model presented in this section supports both clients initiating connections to servers, as well as clients listening for connections from servers calling home.

This model supports both the SSH and TLS transport protocols, using the SSH client and TLS client groupings defined in [I-D.ietf-netconf-ssh-client-server] and [I-D.ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF client supports.

2.1. Tree Diagram

Just the container is displayed below, but there is also a grouping that the container is using.

Note: all lines are folded at column 71 with no '\ ' character.

```

module: ietf-netconf-client
  +--rw netconf-client
    +--rw initiate {initiate}?
      +--rw netconf-server* [name]
        +--rw name string
        +--rw (transport)
          +--:(ssh) {ssh-initiate}?
            +--rw ssh
              +--rw endpoints
                +--rw endpoint* [name]
                  +--rw name string
                  +--rw address inet:host
                  +--rw port? inet:port-number
              +--rw server-auth
                +--rw trusted-ssh-host-keys?
                |   -> /ks:keystore/trusted-host-keys/name
                +--rw trusted-ca-certs? leafref
                |   {sshcom:ssh-x509-certs}?
                +--rw trusted-server-certs? leafref
                |   {sshcom:ssh-x509-certs}?
              +--rw client-auth
                +--rw username? string
                +--rw (auth-type)
                  +--:(certificate)
                  |   +--rw certificate? leafref
                  |   |   {sshcom:ssh-x509-certs}?
                  +--:(public-key)
                  |   +--rw public-key?
                  |   |   -> /ks:keystore/keys/key/name
                  +--:(password)

```



```

|         |--rw password?      string
+--rw transport-params
|   {ssh-client-transport-params-config}?
|   |--rw host-key
|   |   |--rw host-key-alg*    identityref
+--rw key-exchange
|   |--rw key-exchange-alg*    identityref
+--rw encryption
|   |--rw encryption-alg*     identityref
+--rw mac
|   |--rw mac-alg*            identityref
+--rw compression
|   |--rw compression-alg*    identityref
+--:(tls) {tls-initiate}?
+--rw tls
+--rw endpoints
|   |--rw endpoint* [name]
|   |   |--rw name            string
|   |   |--rw address         inet:host
|   |   |--rw port?           inet:port-number
+--rw server-auth
|   |--rw trusted-ca-certs?    leafref
|   |--rw trusted-server-certs? leafref
+--rw client-auth
|   |--rw (auth-type)
|   |   +--:(certificate)
|   |   |--rw certificate?    leafref
+--rw hello-params
|   {tls-client-hello-params-config}?
|   |--rw tls-versions
|   |   |--rw tls-version*    identityref
+--rw cipher-suites
|   |--rw cipher-suite*       identityref
+--rw connection-type
+--rw (connection-type)?
+--:(persistent-connection)
|   |--rw persistent!
|   |   |--rw idle-timeout?    uint32
|   |   |--rw keep-alives
|   |   |   |--rw max-wait?      uint16
|   |   |   |--rw max-attempts? uint8
+--:(periodic-connection)
|   |--rw periodic!
|   |   |--rw idle-timeout?      uint16
|   |   |--rw reconnect-timeout? uint16
+--rw reconnect-strategy
|   |--rw start-with?          enumeration
|   |--rw max-attempts?        uint8

```

```

+--rw listen {listen}?
  +--rw max-sessions?   uint16
  +--rw idle-timeout?   uint16
  +--rw endpoint* [name]
    +--rw name          string
    +--rw (transport)
      +--:(ssh) {ssh-listen}?
        +--rw ssh
          +--rw address?          inet:ip-address
          +--rw port?             inet:port-number
          +--rw server-auth
            +--rw trusted-ssh-host-keys?
              | -> /ks:keystore/trusted-host-keys/name
            +--rw trusted-ca-certs?   leafref
              | {sshcom:ssh-x509-certs}?
            +--rw trusted-server-certs? leafref
              | {sshcom:ssh-x509-certs}?
          +--rw client-auth
            +--rw username?         string
            +--rw (auth-type)
              +--:(certificate)
                | +--rw certificate? leafref
                  | {sshcom:ssh-x509-certs}?
              +--:(public-key)
                | +--rw public-key?
                  | -> /ks:keystore/keys/key/name
              +--:(password)
                | +--rw password?    string
          +--rw transport-params
            {ssh-client-transport-params-config}?
            +--rw host-key
              | +--rw host-key-alg*   identityref
            +--rw key-exchange
              | +--rw key-exchange-alg* identityref
            +--rw encryption
              | +--rw encryption-alg* identityref
            +--rw mac
              | +--rw mac-alg*       identityref
            +--rw compression
              | +--rw compression-alg* identityref
      +--:(tls) {tls-listen}?
        +--rw tls
          +--rw address?          inet:ip-address
          +--rw port?             inet:port-number
          +--rw server-auth
            | +--rw trusted-ca-certs?   leafref
            | +--rw trusted-server-certs? leafref
          +--rw client-auth

```

```
|   +--rw (auth-type)
|   |   +---:(certificate)
|   |   |   +--rw certificate?   leafref
+--rw hello-params
|   {tls-client-hello-params-config}?
+--rw tls-versions
|   +--rw tls-version*   identityref
+--rw cipher-suites
|   +--rw cipher-suite*   identityref
```

2.2. Example Usage

The following example illustrates configuring a NETCONF client to initiate connections, using both the SSH and TLS transport protocols, as well as listening for call-home connections, again using both the SSH and TLS transport protocols.

This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

```
<netconf-client
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-client">

  <!-- NETCONF servers to initiate connections to -->
  <initiate>
    <netconf-server>
      <name>corp-fw1</name>
      <ssh>
        <endpoints>
          <endpoint>
            <name>corp-fw1.example.com</name>
            <address>corp-fw1.example.com</address>
          </endpoint>
          <endpoint>
            <name>corp-fw2.example.com</name>
            <address>corp-fw2.example.com</address>
          </endpoint>
        </endpoints>
        <server-auth>
          <trusted-server-certs>deployment-specific-ca-certs</trusted-server-cer
ts>
        </server-auth>
        <client-auth>
          <username>foobar</username>
          <public-key>ex-rsa-key</public-key>
        </client-auth>
      </ssh>
    </netconf-server>
  </initiate>

  <!-- endpoints to listen for NETCONF Call Home connections on -->
  <listen>
    <endpoint>
      <name>Intranet-facing listener</name>
      <ssh>
        <address>11.22.33.44</address>
        <server-auth>
          <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
          <trusted-server-certs>explicitly-trusted-server-certs</trusted-server-
certs>
          <trusted-ssh-host-keys>explicitly-trusted-ssh-host-keys</trusted-ssh-h
ost-keys>
        </server-auth>
        <client-auth>
          <username>foobar</username>
          <public-key>ex-rsa-key</public-key>
        </client-auth>
      </ssh>
    </endpoint>
  </listen>
</netconf-client>
```

2.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

```
<CODE BEGINS> file "ietf-netconf-client@2017-07-03.yang"

module ietf-netconf-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-client";
  prefix "ncc";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-ssh-client {
    prefix ss;
    revision-date 2017-06-13; // stable grouping definitions
    reference
      "RFC YYYY: SSH Client and Server Models";
  }

  import ietf-tls-client {
    prefix ts;
    revision-date 2017-06-13; // stable grouping definitions
    reference
      "RFC ZZZZ: TLS Client and Server Models";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>

    Author:    Kent Watsen
               <mailto:kwatsen@juniper.net>

    Author:    Gary Wu
               <mailto:garywu@cisco.com>";

  description
```

"This module contains a collection of YANG definitions for configuring NETCONF clients.

Copyright (c) 2014 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

```
revision "2017-07-03" {
  description
    "Initial version";
  reference
    "RFC XXXX: NETCONF Client and Server Models";
}

// Features

feature initiate {
  description
    "The 'initiate' feature indicates that the NETCONF client
    supports initiating NETCONF connections to NETCONF servers
    using at least one transport (e.g., SSH, TLS, etc.).";
}

feature ssh-initiate {
  description
    "The 'ssh-initiate' feature indicates that the NETCONF client
    supports initiating SSH connections to NETCONF servers.";
  reference
    "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature tls-initiate {
  description
    "The 'tls-initiate' feature indicates that the NETCONF client
    supports initiating TLS connections to NETCONF servers.";
  reference
    "RFC 7589: Using the NETCONF Protocol over Transport
    Layer Security (TLS) with Mutual X.509
    Authentication";
}
```

```
}

feature listen {
  description
    "The 'listen' feature indicates that the NETCONF client
    supports opening a port to accept NETCONF server call
    home connections using at least one transport (e.g.,
    SSH, TLS, etc.).";
}

feature ssh-listen {
  description
    "The 'ssh-listen' feature indicates that the NETCONF client
    supports opening a port to listen for incoming NETCONF
    server call-home SSH connections.";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-listen {
  description
    "The 'tls-listen' feature indicates that the NETCONF client
    supports opening a port to listen for incoming NETCONF
    server call-home TLS connections.";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

container netconf-client {
  uses netconf-client;
  description
    "Top-level container for NETCONF client configuration.";
}

grouping netconf-client {
  description
    "Top-level grouping for NETCONF client configuration.";

  container initiate {
    if-feature initiate;
    description
      "Configures client initiating underlying TCP connections.";
    list netconf-server {
      key name;
      description
        "List of NETCONF servers the NETCONF client is to initiate
        connections to.";
      leaf name {
```

```
    type string;
    description
        "An arbitrary name for the NETCONF server.";
}
choice transport {
    mandatory true;
    description
        "Selects between available transports.";

    case ssh {
        if-feature ssh-initiate;
        container ssh {
            description
                "Specifies SSH-specific transport configuration.";
            uses endpoints-container {
                refine endpoints/endpoint/port {
                    default 830;
                }
            }
            uses ss:ssh-client-grouping;
        }
    } // end ssh

    case tls {
        if-feature tls-initiate;
        container tls {
            description
                "Specifies TLS-specific transport configuration.";
            uses endpoints-container {
                refine endpoints/endpoint/port {
                    default 6513;
                }
            }
            uses ts:tls-client-grouping {
                refine "client-auth/auth-type" {
                    mandatory true;
                    description
                        "NETCONF/TLS clients MUST pass some authentication
                        credentials.";
                }
            }
        }
    }
} // end transport

container connection-type {
    description
```



```
"Indicates the kind of connection to use.";
choice connection-type {
  description
    "Selects between available connection types.";
  case persistent-connection {
    container persistent {
      presence true;
      description
        "Maintain a persistent connection to the NETCONF
        server. If the connection goes down, immediately
        start trying to reconnect to it, using the
        reconnection strategy.

        This connection type minimizes any NETCONF server
        to NETCONF client data-transfer delay, albeit at
        the expense of holding resources longer.";
    leaf idle-timeout {
      type uint32;
      units "seconds";
      default 86400; // one day;
      description
        "Specifies the maximum number of seconds that a
        a NETCONF session may remain idle. A NETCONF
        session will be dropped if it is idle for an
        interval longer than this number of seconds.
        If set to zero, then the client will never drop
        a session because it is idle. Sessions that
        have a notification subscription active are
        never dropped.";
    }
    container keep-alives {
      description
        "Configures the keep-alive policy, to proactively
        test the aliveness of the SSH/TLS server. An
        unresponsive SSH/TLS server will be dropped after
        approximately max-attempts * max-wait seconds.";
      reference
        "RFC 8071: NETCONF Call Home and RESTCONF Call
        Home, Section 3.1, item S6";
      leaf max-wait {
        type uint16 {
          range "1..max";
        }
        units seconds;
        default 30;
        description
          "Sets the amount of time in seconds after which
          if no data has been received from the SSH/TLS
```

```
        server, a SSH/TLS-level message will be sent
        to test the aliveness of the SSH/TLS server.";
    }
    leaf max-attempts {
        type uint8;
        default 3;
        description
            "Sets the maximum number of sequential keep-alive
            messages that can fail to obtain a response from
            the SSH/TLS server before assuming the SSH/TLS
            server is no longer alive.";
    }
}
}
}
case periodic-connection {
    container periodic {
        presence true;
        description
            "Periodically connect to the NETCONF server, so that
            the NETCONF server may deliver messages pending for
            the NETCONF client. The NETCONF server must close
            the connection when it is ready to release it. Once
            the connection has been closed, the NETCONF client
            will restart its timer until the next connection.";
        leaf idle-timeout {
            type uint16;
            units "seconds";
            default 300; // five minutes
            description
                "Specifies the maximum number of seconds that a
                a NETCONF session may remain idle. A NETCONF
                session will be dropped if it is idle for an
                interval longer than this number of seconds.
                If set to zero, then the server will never drop
                a session because it is idle. Sessions that
                have a notification subscription active are
                never dropped.";
        }
        leaf reconnect-timeout {
            type uint16 {
                range "1..max";
            }
            units minutes;
            default 60;
            description
                "Sets the maximum amount of unconnected time the
                NETCONF client will wait before re-establishing
```

```

        a connection to the NETCONF server. The NETCONF
        client may initiate a connection before this
        time if desired (e.g., to set configuration).";
    }
}
}
}
container reconnect-strategy {
    description
    "The reconnection strategy directs how a NETCONF client
    reconnects to a NETCONF server, after discovering its
    connection to the server has dropped, even if due to a
    reboot. The NETCONF client starts with the specified
    endpoint and tries to connect to it max-attempts times
    before trying the next endpoint in the list (round
    robin).";
    leaf start-with {
        type enumeration {
            enum first-listed {
                description
                "Indicates that reconections should start with
                the first endpoint listed.";
            }
            enum last-connected {
                description
                "Indicates that reconections should start with
                the endpoint last connected to. If no previous
                connection has ever been established, then the
                first endpoint configured is used. NETCONF
                clients SHOULD be able to remember the last
                endpoint connected to across reboots.";
            }
        }
    }
    default first-listed;
    description
    "Specifies which of the NETCONF server's endpoints the
    NETCONF client should start with when trying to connect
    to the NETCONF server.";
}
leaf max-attempts {
    type uint8 {
        range "1..max";
    }
    default 3;
    description
    "Specifies the number times the NETCONF client tries to
    connect to a specific endpoint before moving on to the
```

```
        next endpoint in the list (round robin).";
    }
} // end netconf-server
} // end initiate

container listen {
  if-feature listen;
  description
    "Configures client accepting call-home TCP connections.";

  leaf max-sessions {
    type uint16;
    default 0;
    description
      "Specifies the maximum number of concurrent sessions
       that can be active at one time. The value 0 indicates
       that no artificial session limit should be used.";
  }

  leaf idle-timeout {
    type uint16;
    units "seconds";
    default 3600; // one hour
    description
      "Specifies the maximum number of seconds that a NETCONF
       session may remain idle. A NETCONF session will be dropped
       if it is idle for an interval longer than this number of
       seconds. If set to zero, then the server will never drop
       a session because it is idle. Sessions that have a
       notification subscription active are never dropped.";
  }

  list endpoint {
    key name;
    description
      "List of endpoints to listen for NETCONF connections.";
    leaf name {
      type string;
      description
        "An arbitrary name for the NETCONF listen endpoint.";
    }
    choice transport {
      mandatory true;
      description
        "Selects between available transports.";
      case ssh {
        if-feature ssh-listen;
      }
    }
  }
}
```

```
    container ssh {
      description
        "SSH-specific listening configuration for inbound
        connections.";
      leaf address {
        type inet:ip-address;
        description
          "The IP address to listen for call-home connections.";
      }
      leaf port {
        type inet:port-number;
        default 4334;
        description
          "The port number to listen for call-home connections.";
      }
      uses ss:ssh-client-grouping;
    }
  }
  case tls {
    if-feature tls-listen;
    container tls {
      description
        "TLS-specific listening configuration for inbound
        connections.";
      leaf address {
        type inet:ip-address;
        description
          "The IP address to listen for call-home connections.";
      }
      leaf port {
        type inet:port-number;
        default 4335;
        description
          "The port number to listen for call-home connections.";
      }
      uses ts:tls-client-grouping {
        refine "client-auth/auth-type" {
          mandatory true;
          description
            "NETCONF/TLS clients MUST pass some authentication
            credentials.";
        }
      }
    }
  }
} // end transport
} // end endpoint
} // end listen
```

```

} // end netconf-client

grouping endpoints-container {
  description
    "This grouping is used to configure a set of NETCONF servers
    a NETCONF client may initiate connections to.";
  container endpoints {
    description
      "Container for the list of endpoints.";
    list endpoint {
      key name;
      unique "address port";
      min-elements 1;
      ordered-by user;
      description
        "A non-empty user-ordered list of endpoints for this NETCONF
        client to try to connect to. Defining more than one enables
        high-availability.";
      leaf name {
        type string;
        description
          "An arbitrary name for this endpoint.";
      }
      leaf address {
        type inet:host;
        mandatory true;
        description
          "The IP address or hostname of the endpoint. If a
          hostname is configured and the DNS resolution results
          in more than one IP address, the NETCONF client
          will process the IP addresses as if they had been
          explicitly configured in place of the hostname.";
      }
      leaf port {
        type inet:port-number;
        description
          "The IP port for this endpoint. The NETCONF client will
          use the IANA-assigned well-known port (set via a refine
          statement when uses) if no value is specified.";
      }
    }
  }
}

```

<CODE ENDS>

3. The NETCONF Server Model

The NETCONF server model presented in this section supports servers both listening for connections as well as initiating call-home connections.

This model supports both the SSH and TLS transport protocols, using the SSH server and TLS server groupings defined in [I-D.ietf-netconf-ssh-client-server] and [I-D.ietf-netconf-tls-client-server] respectively.

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the NETCONF server supports.

3.1. Tree Diagram

Just the container is displayed below, but there is also a grouping that the container is using.

Note: all lines are folded at column 71 with no '\' character.

```

module: ietf-netconf-server
  +--rw netconf-server
    +--rw session-options
      |   +--rw hello-timeout?   uint16
    +--rw listen {listen}?
      |   +--rw max-sessions?    uint16
      |   +--rw idle-timeout?   uint16
      |   +--rw endpoint* [name]
      |     +--rw name          string
      |     +--rw (transport)
      |       +--:(ssh) {ssh-listen}?
      |         +--rw ssh
      |           +--rw address?          inet:ip-address
      |           +--rw port?             inet:port-number
      |           +--rw host-keys
      |             +--rw host-key* [name]
      |               +--rw name          string
      |               +--rw (host-key-type)
      |                 +--:(public-key)
      |                   +--rw public-key?
      |                     -> /ks:keystore/keys/key/name
      |                 +--:(certificate)
      |                   +--rw certificate?  leafref
      |                     {sshcom:ssh-x509-certs}?

```

```

+--rw client-cert-auth {sshcom:ssh-x509-certs}?
|   +--rw trusted-ca-certs?      leafref
|   +--rw trusted-client-certs?  leafref
+--rw transport-params
|   {ssh-server-transport-params-config}?
|   +--rw host-key
|   |   +--rw host-key-alg*      identityref
+--rw key-exchange
|   +--rw key-exchange-alg*      identityref
+--rw encryption
|   +--rw encryption-alg*        identityref
+--rw mac
|   +--rw mac-alg*               identityref
+--rw compression
|   +--rw compression-alg*       identityref
+--:(tls) {tls-listen}?
+--rw tls
|   +--rw address?               inet:ip-address
|   +--rw port?                  inet:port-number
|   +--rw certificates
|   |   +--rw certificate* [name]
|   |   |   +--rw name          leafref
+--rw client-auth
|   +--rw trusted-ca-certs?      leafref
|   +--rw trusted-client-certs? leafref
|   +--rw cert-maps
|   |   +--rw cert-to-name* [id]
|   |   |   +--rw id            uint32
|   |   |   +--rw fingerprint   x509c2n:tls-fingerprint
|   |   |   +--rw map-type       identityref
|   |   |   +--rw name          string
+--rw hello-params
|   {tls-server-hello-params-config}?
|   +--rw tls-versions
|   |   +--rw tls-version*       identityref
+--rw cipher-suites
|   +--rw cipher-suite*          identityref
+--rw call-home {call-home}?
+--rw netconf-client* [name]
|   +--rw name                    string
+--rw (transport)
|   +--:(ssh) {ssh-call-home}?
|   |   +--rw ssh
|   |   |   +--rw endpoints
|   |   |   |   +--rw endpoint* [name]
|   |   |   |   |   +--rw name          string
|   |   |   |   |   +--rw address       inet:host
|   |   |   |   |   +--rw port?         inet:port-number

```



```

+--rw host-keys
|   +--rw host-key* [name]
|       +--rw name          string
|       +--rw (host-key-type)
|           +--:(public-key)
|               +--rw public-key?
|                   -> /ks:keystore/keys/key/name
|           +--:(certificate)
|               +--rw certificate? leafref
|                   {sshcom:ssh-x509-certs}?
+--rw client-cert-auth {sshcom:ssh-x509-certs}?
|   +--rw trusted-ca-certs? leafref
|   +--rw trusted-client-certs? leafref
+--rw transport-params
|   {ssh-server-transport-params-config}?
|   +--rw host-key
|       |   +--rw host-key-alg* identityref
+--rw key-exchange
|   |   +--rw key-exchange-alg* identityref
+--rw encryption
|   |   +--rw encryption-alg* identityref
+--rw mac
|   |   +--rw mac-alg* identityref
+--rw compression
|   |   +--rw compression-alg* identityref
+--:(tls) {tls-call-home}?
+--rw tls
+--rw endpoints
|   +--rw endpoint* [name]
|       +--rw name          string
|       +--rw address       inet:host
|       +--rw port?         inet:port-number
+--rw certificates
|   +--rw certificate* [name]
|       +--rw name          leafref
+--rw client-auth
|   +--rw trusted-ca-certs? leafref
|   +--rw trusted-client-certs? leafref
+--rw cert-maps
|   +--rw cert-to-name* [id]
|       +--rw id            uint32
|       +--rw fingerprint   x509c2n:tls-fingerprint
|       +--rw map-type       identityref
|       +--rw name           string
+--rw hello-params
|   {tls-server-hello-params-config}?
+--rw tls-versions
|   +--rw tls-version* identityref

```

```

|           +--rw cipher-suites
|           +--rw cipher-suite*  identityref
+--rw connection-type
|   +--rw (connection-type)?
|   |   +--:(persistent-connection)
|   |   |   +--rw persistent!
|   |   |   |   +--rw idle-timeout?    uint32
|   |   |   |   +--rw keep-alives
|   |   |   |   |   +--rw max-wait?      uint16
|   |   |   |   |   +--rw max-attempts?  uint8
|   |   +--:(periodic-connection)
|   |   |   +--rw periodic!
|   |   |   |   +--rw idle-timeout?      uint16
|   |   |   |   +--rw reconnect-timeout?  uint16
+--rw reconnect-strategy
|   +--rw start-with?    enumeration
|   +--rw max-attempts?  uint8

```

3.2. Example Usage

The following example illustrates configuring a NETCONF server to listen for NETCONF client connections using both the SSH and TLS transport protocols, as well as configuring call-home to two NETCONF clients, one using SSH and the other using TLS.

This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

```

<netconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server"
  xmlns:x509c2n="urn:ietf:params:xml:ns:yang:ietf-x509-cert-to-name">

  <!-- listening for SSH and TLS connections -->
  <listen>
    <endpoint> <!-- listening for SSH connections -->
      <name>netconf/ssh</name>
      <ssh>
        <address>11.22.33.44</address>
        <host-keys>
          <host-key>
            <name>public-key</name>
            <public-key>ex-rsa-key</public-key>
          </host-key>
          <host-key>
            <name>certificate</name>
            <certificate>builtin-idevid-cert</certificate>
          </host-key>
        </host-keys>
      </ssh>
    </endpoint>
  </listen>

```

```

        <client-cert-auth>
          <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
          <trusted-client-certs>explicitly-trusted-client-certs</trusted-client-
certs>
        </client-cert-auth>
      </ssh>
    </endpoint>
    <endpoint> <!-- listening for TLS sessions -->
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>
            <name>tls-ec-cert</name>
          </certificate>
        </certificates>
        <client-auth>
          <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
          <trusted-client-certs>explicitly-trusted-client-certs</trusted-client-
certs>
        <cert-maps>
          <cert-to-name>
            <id>1</id>
            <fingerprint>11:0A:05:11:00</fingerprint>
            <map-type>x509c2n:san-any</map-type>
          </cert-to-name>
          <cert-to-name>
            <id>2</id>
            <fingerprint>B3:4F:A1:8C:54</fingerprint>
            <map-type>x509c2n:specified</map-type>
            <name>scooby-doo</name>
          </cert-to-name>
        </cert-maps>
      </client-auth>
    </tls>
  </endpoint>
</listen>

<!-- calling home to an SSH and TLS based NETCONF clients -->
<call-home>
  <netconf-client> <!-- SSH-based client -->
    <name>config-mgr</name>
    <ssh>
      <endpoints>
        <endpoint>
          <name>east-data-center</name>
          <address>11.22.33.44</address>
        </endpoint>
        <endpoint>
          <name>west-data-center</name>

```

```

        <address>55.66.77.88</address>
      </endpoint>
    </endpoints>
  <host-keys>
    <host-key>
      <name>certificate</name>
      <certificate>builtin-idevid-cert</certificate>
    </host-key>
  </host-keys>
  <client-cert-auth>
    <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
    <trusted-client-certs>explicitly-trusted-client-certs</trusted-client-
certs>
  </client-cert-auth>
</ssh>
<connection-type>
  <periodic>
    <idle-timeout>300</idle-timeout>
    <reconnect-timeout>60</reconnect-timeout>
  </periodic>
</connection-type>
<reconnect-strategy>
  <start-with>last-connected</start-with>
  <max-attempts>3</max-attempts>
</reconnect-strategy>
</netconf-client>
<netconf-client> <!-- TLS-based client -->
  <name>event-correlator</name>
  <tls>
    <endpoints>
      <endpoint>
        <name>east-data-center</name>
        <address>22.33.44.55</address>
      </endpoint>
      <endpoint>
        <name>west-data-center</name>
        <address>33.44.55.66</address>
      </endpoint>
    </endpoints>
    <certificates>
      <certificate>
        <name>tls-ec-cert</name>
      </certificate>
    </certificates>
    <client-auth>
      <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
      <trusted-client-certs>explicitly-trusted-client-certs</trusted-client-
certs>
    <cert-maps>
      <cert-to-name>

```

```

        <id>1</id>
        <fingerprint>11:0A:05:11:00</fingerprint>
        <map-type>x509c2n:san-any</map-type>
      </cert-to-name>
      <cert-to-name>
        <id>2</id>
        <fingerprint>B3:4F:A1:8C:54</fingerprint>
        <map-type>x509c2n:specified</map-type>
        <name>scooby-doo</name>
      </cert-to-name>
    </cert-maps>
  </client-auth>
</tls>
<connection-type>
  <persistent>
    <idle-timeout>300</idle-timeout>
    <keep-alives>
      <max-wait>30</max-wait>
      <max-attempts>3</max-attempts>
    </keep-alives>
  </persistent>
</connection-type>
<reconnect-strategy>
  <start-with>first-listed</start-with>
  <max-attempts>3</max-attempts>
</reconnect-strategy>
</netconf-client>
</call-home>
</netconf-server>

```

3.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

```

<CODE BEGINS> file "ietf-netconf-server@2017-07-03.yang"

module ietf-netconf-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-server";
  prefix "ncs";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
}

```

```
import ietf-x509-cert-to-name {
  prefix x509c2n;
  reference
    "RFC 7407: A YANG Data Model for SNMP Configuration";
}

import ietf-ssh-server {
  prefix ss;
  revision-date 2017-06-13; // stable grouping definitions
  reference
    "RFC YYYY: SSH Client and Server Models";
}

import ietf-tls-server {
  prefix ts;
  revision-date 2017-06-13; // stable grouping definitions
  reference
    "RFC ZZZZ: TLS Client and Server Models";
}

organization
  "IETF NETCONF (Network Configuration) Working Group";

contact
  "WG Web:    <http://tools.ietf.org/wg/netconf/>
  WG List:    <mailto:netconf@ietf.org>

  Author:     Kent Watsen
               <mailto:kwatsen@juniper.net>";

description
  "This module contains a collection of YANG definitions for
  configuring NETCONF servers.

  Copyright (c) 2014 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD
  License set forth in Section 4.c of the IETF Trust's
  Legal Provisions Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";
```

```
revision "2017-07-03" {
  description
    "Initial version";
  reference
    "RFC XXXX: NETCONF Client and Server Models";
}

// Features

feature listen {
  description
    "The 'listen' feature indicates that the NETCONF server
    supports opening a port to accept NETCONF client connections
    using at least one transport (e.g., SSH, TLS, etc.).";
}

feature ssh-listen {
  description
    "The 'ssh-listen' feature indicates that the NETCONF server
    supports opening a port to accept NETCONF over SSH
    client connections.";
  reference
    "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature tls-listen {
  description
    "The 'tls-listen' feature indicates that the NETCONF server
    supports opening a port to accept NETCONF over TLS
    client connections.";
  reference
    "RFC 7589: Using the NETCONF Protocol over Transport
    Layer Security (TLS) with Mutual X.509
    Authentication";
}

feature call-home {
  description
    "The 'call-home' feature indicates that the NETCONF server
    supports initiating NETCONF call home connections to NETCONF
    clients using at least one transport (e.g., SSH, TLS, etc.).";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature ssh-call-home {
  description
```

```
    "The 'ssh-call-home' feature indicates that the NETCONF
    server supports initiating a NETCONF over SSH call
    home connection to NETCONF clients.";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-call-home {
  description
    "The 'tls-call-home' feature indicates that the NETCONF
    server supports initiating a NETCONF over TLS call
    home connection to NETCONF clients.";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

container netconf-server {
  uses netconf-server;
  description
    "Top-level container for NETCONF server configuration.";
}

grouping netconf-server {
  description
    "Top-level grouping for NETCONF server configuration.";

  container session-options { // SHOULD WE REMOVE THIS ALTOGETHER?
    description
      "NETCONF session options, independent of transport
      or connection strategy.";
    leaf hello-timeout {
      type uint16;
      units "seconds";
      default 600;
      description
        "Specifies the maximum number of seconds that a SSH/TLS
        connection may wait for a hello message to be received.
        A connection will be dropped if no hello message is
        received before this number of seconds elapses. If set
        to zero, then the server will wait forever for a hello
        message.";
    }
  }
}

container listen {
  if-feature listen;
  description
    "Configures listen behavior";
}
```



```
leaf max-sessions {
  type uint16;
  default 0;
  description
    "Specifies the maximum number of concurrent sessions
     that can be active at one time. The value 0 indicates
     that no artificial session limit should be used.";
}
leaf idle-timeout {
  type uint16;
  units "seconds";
  default 3600; // one hour
  description
    "Specifies the maximum number of seconds that a NETCONF
     session may remain idle. A NETCONF session will be dropped
     if it is idle for an interval longer than this number of
     seconds. If set to zero, then the server will never drop
     a session because it is idle. Sessions that have a
     notification subscription active are never dropped.";
}
list endpoint {
  key name;
  description
    "List of endpoints to listen for NETCONF connections.";
  leaf name {
    type string;
    description
      "An arbitrary name for the NETCONF listen endpoint.";
  }
}

choice transport {
  mandatory true;
  description
    "Selects between available transports.";
  case ssh {
    if-feature ssh-listen;
    container ssh {
      description
        "SSH-specific listening configuration for inbound
         connections.";
      leaf address {
        type inet:ip-address;
        description
          "The IP address of the interface to listen on. The
           SSH server will listen on all interfaces if no value
           is specified. Please note that some addresses have
           special meanings (e.g., '0.0.0.0' and ':::').";
      }
    }
  }
}
```

```

    }
    leaf port {
        type inet:port-number;
        default 830;
        description
            "The local port number on this interface the SSH server
            listens on.";
    }
    uses ss:ssh-server-grouping;
}
}
case tls {
    if-feature tls-listen;
    container tls {
        description
            "TLS-specific listening configuration for inbound
            connections.";
        leaf address {
            type inet:ip-address;
            description
                "The IP address of the interface to listen on. The
                TLS server will listen on all interfaces if no value
                is specified. Please note that some addresses have
                special meanings (e.g., '0.0.0.0' and '::').";
        }
        leaf port {
            type inet:port-number;
            default 6513;
            description
                "The local port number on this interface the TLS server
                listens on.";
        }
        uses ts:tls-server-grouping {
            refine "client-auth" {
                must 'trusted-ca-certs or trusted-client-certs';
                description
                    "NETCONF/TLS servers MUST validate client
                    certificates.";
            }
            augment "client-auth" {
                description
                    "Augments in the cert-to-name structure.";
                uses cert-maps-grouping;
            }
        }
    }
}
}
}

```

```
    }  
  }  
  
  container call-home {  
    if-feature call-home;  
    description  
      "Configures call-home behavior";  
    list netconf-client {  
      key name;  
      description  
        "List of NETCONF clients the NETCONF server is to initiate  
        call-home connections to.";  
      leaf name {  
        type string;  
        description  
          "An arbitrary name for the remote NETCONF client.";  
      }  
      choice transport {  
        mandatory true;  
        description  
          "Selects between available transports.";  
        case ssh {  
          if-feature ssh-call-home;  
          container ssh {  
            description  
              "Specifies SSH-specific call-home transport  
              configuration.";  
            uses endpoints-container {  
              refine endpoints/endpoint/port {  
                default 4334;  
              }  
            }  
            uses ss:ssh-server-grouping;  
          }  
        }  
        case tls {  
          if-feature tls-call-home;  
          container tls {  
            description  
              "Specifies TLS-specific call-home transport  
              configuration.";  
            uses endpoints-container {  
              refine endpoints/endpoint/port {  
                default 4335;  
              }  
            }  
            uses ts:tls-server-grouping {  
              refine "client-auth" {
```

```

    must 'trusted-ca-certs or trusted-client-certs';
    description
        "NETCONF/TLS servers MUST validate client
        certificates.";
}
augment "client-auth" {
    description
        "Augments in the cert-to-name structure.";
    uses cert-maps-grouping;
}
}
}
}
}
container connection-type {
    description
        "Indicates the kind of connection to use.";
    choice connection-type {
        description
            "Selects between available connection types.";
        case persistent-connection {
            container persistent {
                presence true;
                description
                    "Maintain a persistent connection to the NETCONF
                    client. If the connection goes down, immediately
                    start trying to reconnect to it, using the
                    reconnection strategy."

                This connection type minimizes any NETCONF client
                to NETCONF server data-transfer delay, albeit at
                the expense of holding resources longer.";
            leaf idle-timeout {
                type uint32;
                units "seconds";
                default 86400; // one day;
                description
                    "Specifies the maximum number of seconds that a
                    a NETCONF session may remain idle. A NETCONF
                    session will be dropped if it is idle for an
                    interval longer than this number of seconds.
                    If set to zero, then the server will never drop
                    a session because it is idle. Sessions that
                    have a notification subscription active are
                    never dropped.";
            }
        }
    }
    container keep-alives {
        description

```

```
        "Configures the keep-alive policy, to proactively
        test the aliveness of the SSH/TLS client.  An
        unresponsive SSH/TLS client will be dropped after
        approximately max-attempts * max-wait seconds.";
reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call
    Home, Section 3.1, item S6";
leaf max-wait {
    type uint16 {
        range "1..max";
    }
    units seconds;
    default 30;
    description
        "Sets the amount of time in seconds after which
        if no data has been received from the SSH/TLS
        client, a SSH/TLS-level message will be sent
        to test the aliveness of the SSH/TLS client.";
}
leaf max-attempts {
    type uint8;
    default 3;
    description
        "Sets the maximum number of sequential keep-alive
        messages that can fail to obtain a response from
        the SSH/TLS client before assuming the SSH/TLS
        client is no longer alive.";
}
}
}
}
}
case periodic-connection {
    container periodic {
        presence true;
        description
            "Periodically connect to the NETCONF client, so that
            the NETCONF client may deliver messages pending for
            the NETCONF server.  The NETCONF client must close
            the connection when it is ready to release it.  Once
            the connection has been closed, the NETCONF server
            will restart its timer until the next connection.";
        leaf idle-timeout {
            type uint16;
            units "seconds";
            default 300; // five minutes
            description
                "Specifies the maximum number of seconds that a
                a NETCONF session may remain idle.  A NETCONF
```

```

    session will be dropped if it is idle for an interval longer than this number of seconds. If set to zero, then the server will never drop a session because it is idle. Sessions that have a notification subscription active are never dropped.";
}
leaf reconnect-timeout {
    type uint16 {
        range "1..max";
    }
    units minutes;
    default 60;
    description
        "Sets the maximum amount of unconnected time the NETCONF server will wait before re-establishing a connection to the NETCONF client. The NETCONF server may initiate a connection before this time if desired (e.g., to deliver an event notification message).";
}
}
}
}
}
container reconnect-strategy {
    description
        "The reconnection strategy directs how a NETCONF server reconnects to a NETCONF client, after discovering its connection to the client has dropped, even if due to a reboot. The NETCONF server starts with the specified endpoint and tries to connect to it max-attempts times before trying the next endpoint in the list (round robin).";
    leaf start-with {
        type enumeration {
            enum first-listed {
                description
                    "Indicates that reconconnections should start with the first endpoint listed.";
            }
            enum last-connected {
                description
                    "Indicates that reconconnections should start with the endpoint last connected to. If no previous connection has ever been established, then the first endpoint configured is used. NETCONF servers SHOULD be able to remember the last

```

```

        endpoint connected to across reboots.";
    }
}
default first-listed;
description
    "Specifies which of the NETCONF client's endpoints the
    NETCONF server should start with when trying to connect
    to the NETCONF client.";
}
leaf max-attempts {
    type uint8 {
        range "1..max";
    }
    default 3;
    description
        "Specifies the number times the NETCONF server tries to
        connect to a specific endpoint before moving on to the
        next endpoint in the list (round robin).";
}
}
}
}
}
}

```

```

grouping cert-maps-grouping {
    description
        "A grouping that defines a container around the
        cert-to-name structure defined in RFC 7407.";
    container cert-maps {
        uses x509c2n:cert-to-name;
        description
            "The cert-maps container is used by a TLS-based NETCONF
            server to map the NETCONF client's presented X.509
            certificate to a NETCONF username. If no matching and
            valid cert-to-name list entry can be found, then the
            NETCONF server MUST close the connection, and MUST NOT
            accept NETCONF messages over it.";
        reference
            "RFC WWW: NETCONF over TLS, Section 7";
    }
}

```

```

grouping endpoints-container {
    description
        "This grouping is used to configure a set of NETCONF clients
        a NETCONF server may initiate call-home connections to.";
}

```

```
container endpoints {
  description
    "Container for the list of endpoints.";
  list endpoint {
    key name;
    unique "address port";
    min-elements 1;
    ordered-by user;
    description
      "A non-empty user-ordered list of endpoints for this NETCONF
      server to try to connect to. Defining more than one enables
      high-availability.";
    leaf name {
      type string;
      description
        "An arbitrary name for this endpoint.";
    }
    leaf address {
      type inet:host;
      mandatory true;
      description
        "The IP address or hostname of the endpoint. If a
        hostname is configured and the DNS resolution results
        in more than one IP address, the NETCONF server
        will process the IP addresses as if they had been
        explicitly configured in place of the hostname.";
    }
    leaf port {
      type inet:port-number;
      description
        "The IP port for this endpoint. The NETCONF server will
        use the IANA-assigned well-known port (set via a refine
        statement when uses) if no value is specified.";
    }
  }
}
```

<CODE ENDS>

4. Design Considerations

Editorial: this section is a hold over from before, previously called "Objectives". It was only written to support the "server" (not the

"client"). The question is if it's better to add the missing "client" parts, or remove this section altogether.

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF client and servers. This scope includes the following objectives:

4.1. Support all NETCONF transports

The YANG module should support all current NETCONF transports, namely NETCONF over SSH [RFC6242], NETCONF over TLS [RFC7589], and to be extensible to support future transports as necessary.

Because implementations may not support all transports, the modules should use YANG "feature" statements so that implementations can accurately advertise which transports are supported.

4.2. Enable each transport to select which keys to use

Servers may have a multiplicity of host-keys or server-certificates from which subsets may be selected for specific uses. For instance, a NETCONF server may want to use one set of SSH host-keys when listening on port 830, and a different set of SSH host-keys when calling home. The data models provided herein should enable configuration of which keys to use on a per-use basis.

4.3. Support authenticating NETCONF clients certificates

When a certificate is used to authenticate a NETCONF client, there is a need to configure the server to know how to authenticate the certificates. The server should be able to authenticate the client's certificate either by using path-validation to a configured trust anchor or by matching the client-certificate to one previously configured.

4.4. Support mapping authenticated NETCONF client certificates to usernames

When a client certificate is used for TLS client authentication, the NETCONF server must be able to derive a username from the authenticated certificate. Thus the modules defined herein should enable this mapping to be configured.

4.5. Support both listening for connections and call home

The NETCONF protocols were originally defined as having the server opening a port to listen for client connections. More recently the NETCONF working group defined support for call-home ([RFC8071]),

enabling the server to initiate the connection to the client. Thus the modules defined herein should enable configuration for both listening for connections and calling home. Because implementations may not support both listening for connections and calling home, YANG "feature" statements should be used so that implementation can accurately advertise the connection types it supports.

4.6. For Call Home connections

The following objectives only pertain to call home connections.

4.6.1. Support more than one NETCONF client

A NETCONF server may be managed by more than one NETCONF client. For instance, a deployment may have one client for provisioning and another for fault monitoring. Therefore, when it is desired for a server to initiate call home connections, it should be able to do so to more than one client.

4.6.2. Support NETCONF clients having more than one endpoint

A NETCONF client managing a NETCONF server may implement a high-availability strategy employing a multiplicity of active and/or passive endpoint. Therefore, when it is desired for a server to initiate call home connections, it should be able to connect to any of the client's endpoints.

4.6.3. Support a reconnection strategy

Assuming a NETCONF client has more than one endpoint, then it becomes necessary to configure how a NETCONF server should reconnect to the client should it lose its connection to one the client's endpoints. For instance, the NETCONF server may start with first endpoint defined in a user-ordered list of endpoints or with the last endpoints it was connected to.

4.6.4. Support both persistent and periodic connections

NETCONF clients may vary greatly on how frequently they need to interact with a NETCONF server, how responsive interactions need to be, and how many simultaneous connections they can support. Some clients may need a persistent connection to servers to optimize real-time interactions, while others prefer periodic interactions in order to minimize resource requirements. Therefore, when it is necessary for server to initiate connections, it should be configurable if the connection is persistent or periodic.

4.6.5. Reconnection strategy for periodic connections

The reconnection strategy should apply to both persistent and periodic connections. How it applies to periodic connections becomes clear when considering that a periodic "connection" is a logical connection to a single server. That is, the periods of unconnectedness are intentional as opposed to due to external reasons. A periodic "connection" should always reconnect to the same server until it is no longer able to, at which time the reconnection strategy guides how to connect to another server.

4.6.6. Keep-alives for persistent connections

If a persistent connection is desired, it is the responsibility of the connection initiator to actively test the "aliveness" of the connection. The connection initiator must immediately work to reestablish a persistent connection as soon as the connection is lost. How often the connection should be tested is driven by NETCONF client requirements, and therefore keep-alive settings should be configurable on a per-client basis.

4.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the NETCONF server to know how often it should connect. This frequency determines the maximum amount of time a NETCONF client may have to wait to send data to a server. A server may connect to a client before this interval expires if desired (e.g., to send data to a client).

5. Security Considerations

A denial of service (DoS) attack MAY occur if the NETCONF server limits the maximum number of NETCONF sessions it will accept (i.e. the 'max-sessions' field in the ietf-netconf-server module is not zero) and either the "hello-timeout" or "idle-timeout" fields in ietf-netconf-server module have been set to indicate the NETCONF server should wait forever (i.e. set to zero).

The YANG module defined in this document uses groupings defined in [I-D.ietf-netconf-ssh-client-server] and [I-D.ietf-netconf-tls-client-server]. Please see the Security Considerations section in those documents for concerns related those groupings.

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-

implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

6. IANA Considerations

6.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

6.2. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the the following registrations are requested:

```
name:      ietf-netconf-client
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-client
prefix:    ncc
reference:  RFC XXXX

name:      ietf-netconf-server
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-server
prefix:    ncs
reference:  RFC XXXX
```

7. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, Balazs Kovacs, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Phil Shafer, Sean Turner, and Bert Wijnen.

Juergen Schoenwaelder and was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

8. References

8.1. Normative References

- [I-D.ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-02 (work in progress), June 2017.
- [I-D.ietf-netconf-ssh-client-server]
Watsen, K. and G. Wu, "SSH Client and Server Models", draft-ietf-netconf-ssh-client-server-03 (work in progress), June 2017.
- [I-D.ietf-netconf-tls-client-server]
Watsen, K. and G. Wu, "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-03 (work in progress), June 2017.

- [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>>.
- [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>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <<http://www.rfc-editor.org/info/rfc7407>>.
- [RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <<http://www.rfc-editor.org/info/rfc7589>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.

8.2. Informative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <<http://www.rfc-editor.org/info/rfc4252>>.

- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<http://www.rfc-editor.org/info/rfc4253>>.
- [RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <<http://www.rfc-editor.org/info/rfc4254>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.
- [RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <<http://www.rfc-editor.org/info/rfc8071>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Added in previously missing ietf-netconf-client module.
- o Added in new features 'listen' and 'call-home' so future transports can be augmented in.

A.2. 00 to 01

- o Renamed "keychain" to "keystore".

A.3. 01 to 02

- o Added to ietf-netconf-client ability to connected to a cluster of endpoints, including a reconnection-strategy.
- o Added to ietf-netconf-client the ability to configure connection-type and also keep-alive strategy.
- o Updated both modules to accomodate new groupings in the ssh/tls drafts.

A.4. 02 to 03

- o Refined use of tls-client-grouping to add a must statement indicating that the TLS client must specify a client-certificate.
- o Changed 'netconf-client' to be a grouping (not a container).

A.5. 03 to 04

- o Added RFC 8174 to Requirements Language Section.
- o Replaced refine statement in ietf-netconf-client to add a mandatory true.
- o Added refine statement in ietf-netconf-server to add a must statement.
- o Now there are containers and groupings, for both the client and server models.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

A. Gonzalez Prieto
VMware
A. Clemm
Huawei
E. Voit
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
July 3, 2017

NETCONF Support for Event Notifications
draft-ietf-netconf-netconf-event-notifications-04

Abstract

This document defines how to transport network subscriptions and event messages on top of the Network Configuration protocol (NETCONF). This includes the full set of RPCs, subscription state changes, and message payloads needing asynchronous delivery. The capabilities and operations defined in this document used in conjunction with [subscribe] are intended to obsolete [RFC5277]. In addition, the capabilities within those two documents along with [yang-push] are intended to enable an extract of a YANG datastore on a remote device.

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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 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	3
3. Solution	3
3.1. Event Stream Discovery	3
3.2. Mandatory NETCONF support	4
3.3. Dynamic Subscriptions	5
3.4. Configured Subscriptions	12
4. Interleave Capability	25
5. Security Considerations	26
6. Acknowledgments	26
7. References	27
7.1. Normative References	27
7.2. Informative References	27
Appendix A. Open Items	28
Appendix B. Changes between revisions	28
B.1. v03 to v04	28
B.2. v01 to v03	28
B.3. v00 to v01	28
Authors' Addresses	28

1. Introduction

This document defines mechanisms that provide an asynchronous message notification delivery service for the NETCONF protocol [RFC6241] based on [subscribe]. This is an optional capability built on top of the base NETCONF definition.

The document [subscribe] plus this transport specification document provides a superset of the capabilities previously defined in [RFC5277]. Newly introduced capabilities include the ability to have multiple subscriptions on a single NETCONF session, the ability to terminate subscriptions without terminating the client session, and the ability to modify existing subscriptions.

In addition, [yang-push] plus the capabilities of this document provide a mechanism for a NETCONF client to maintain a subset/extract of an actively changing YANG datastore.

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, server, operation, RPC.

The following terms are defined in [subscribe]: event, event notification, stream, publisher, receiver, subscriber, subscription, configured subscription.

Note that a publisher in [subscribe] corresponds to a server in [RFC6241]. Similarly, a subscriber corresponds to a client. A receiver is also a client. In the remainder of this document, we will use the terminology in [RFC6241] to simplify [subscribe]'s mental mappings to embedded NETCONF terminology.

3. Solution

In this section, we describe and exemplify how [subscribe] is to be supported over NETCONF.

3.1. Event Stream Discovery

In the context of [subscribe] an event stream exposes a continuous set of events available for subscription. A NETCONF client can retrieve the list of available event streams from a NETCONF server using the "get" operation against the top-level container "/streams" defined in [subscribe]. The reply includes the stream identities supported on the NETCONF server.

The following example illustrates the retrieval of the list of available event streams using the <get> operation.

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <streams
        xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0"/>
      </filter>
    </get>
  </rpc>
```

Figure 1: Get streams request

The NETCONF server returns a list of event streams available. In this example, the list contains the NETCONF, SNMP, and SYSLOG streams.

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <streams
      xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
      <stream>NETCONF</stream>
      <stream>SNMP</stream>
      <stream>SYSLOG</stream>
    </streams>
  </data>
</rpc-reply>
```

Figure 2: Get streams response

For [yang-push], a similar get is needed to retrieve available datastore names.

3.2. Mandatory NETCONF support

A NETCONF server implementation supporting [subscribe] must support dynamic subscriptions and the "NETCONF" notification event stream. The NETCONF event stream contains all NETCONF XML event information supported by the server, except for where it has been explicitly indicated that this the event must be excluded from the NETCONF stream.

A NETCONF server implementation supporting [yang-push] must support the "running" datastore.

3.3. Dynamic Subscriptions

3.3.1. Establishing Dynamic Subscriptions

The dynamic subscription RFC and interactions operation is defined in [subscribe].

3.3.1.1. Usage Example

An example of interactions over NETCONF transport for one sample subscription is below:

```
<netconf:rpc netconf:message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <stream>NETCONF</stream>
    <event-filter-type>xpath</event-filter-type>
    <event-filter> xmlns:ex="http://example.com/event/1.0"
      select="/ex:event[ex:eventClass='fault' and
        (ex:severity='minor' or ex:severity='major'
        or ex:severity='critical')]"
    </event-filter>
  </establish-subscription>
</netconf:rpc>
```

Figure 3: establish-subscription over NETCONF

3.3.1.2. Positive Response

If the NETCONF server can satisfy the request, the server sends a positive <subscription-result> element, and the subscription-id of the accepted subscription.

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    ok
  </subscription-result>
  <identifier
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    52
  </identifier>
</rpc-reply>
```

Figure 4: Successful establish-subscription

3.3.1.3. Negative Response

If the NETCONF server cannot satisfy the request, or client has no authorization to establish the subscription, the server will send a negative <subscription-result> element. For instance:

```
<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    stream-unavailable
  </subscription-result>
</rpc-reply>
```

Figure 5: Unsuccessful establish subscription

3.3.1.4. Subscription Negotiation

If the client requests parameters the NETCONF server cannot serve, the negative <subscription-result> may include hints at subscription parameters which would have been accepted. For instance, consider the following subscription from [yang-push], which augments the establish-subscription with some additional parameters, including "period". If the client requests a period which the NETCONF server cannot serve, the back-and-forth exchange may be:

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <stream>push-update</stream>
    <event-filter-type>subtree</event-filter-type>
    <event-filter>xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"</event-filter>
    <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      500
    </period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    error-insufficient-resources
  </subscription-result>
  <period
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    2000
  </period>
</rpc-reply>
```

Figure 6: Subscription establishment negotiation

3.3.1.5. Message Flow Examples

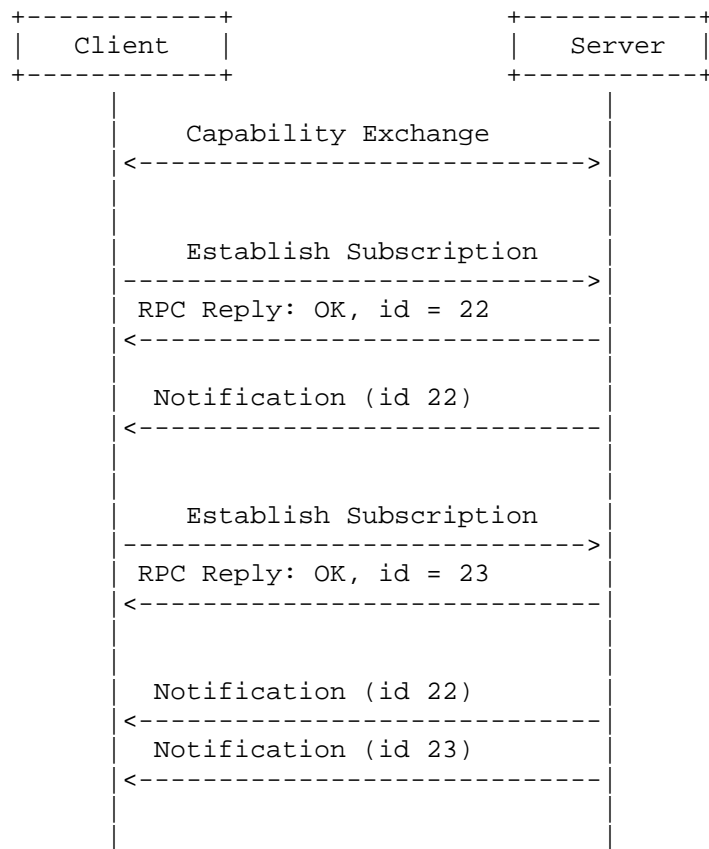


Figure 7: Multiple subscription establishments over a single NETCONF session

3.3.2. Modifying a Subscription

This operation is defined in [subscribe] and enhanced in [yang-push].

3.3.2.1. Usage Example

The following demonstrates modifying a subscription. Consider a subscription from [yang-push], which augments the establish-subscription with some additional parameters, including "period". A subscription may be established and modified as follows.

```

<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <datastore xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">running</d
atastore>
    <event-filter-type>subtree</event-filter-type>
    <event-filter>xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"</event-filter>
    <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
      1000
    </period>
  </establish-subscription>
</netconf:rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    ok
  </subscription-result>
  <identifier
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    1922
  </identifier>
</rpc-reply>

<netconf:rpc message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <identifier>1922</identifier>
    <period>100</period>
  </modify-subscription>
</netconf:rpc>

```

Figure 8: Subscription modification

3.3.2.2. Positive Response

If the NETCONF server can satisfy the request, the server sends a positive `<subscription-result>` element. This response is like that to an establish-subscription request, but without the subscription identifier.

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    ok
  </subscription-result>
</rpc-reply>
```

Figure 9: Successful modify subscription

3.3.2.3. Negative Response

If the NETCONF server cannot satisfy the request, the server sends a negative `<subscription-result>` element. Its contents and semantics are identical to those in an establish-subscription request. For instance:

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    period-unsupported
  </subscription-result>
  <period-hint>500</period-hint>
</rpc-reply>
```

Figure 10: Unsuccessful modify subscription

3.3.2.4. Message Flow Example

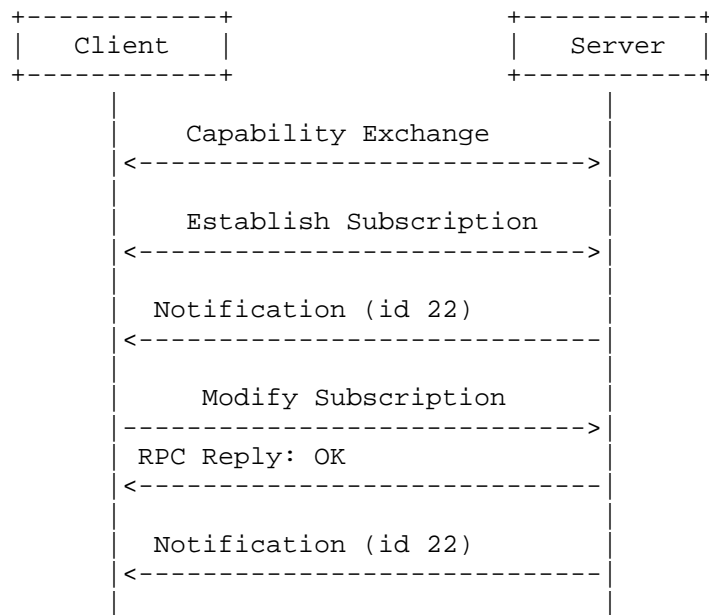


Figure 11: Message flow for successful subscription modification

3.3.3. Deleting a Subscription

This operation is defined in [subscribe] for events, and enhanced in [yang-push] for datastores.

3.3.3.1. Usage Example

The following demonstrates deleting a subscription.

```

<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <identifier>1922</identifier>
  </delete-subscription>
</netconf:rpc>
  
```

Figure 12: Delete subscription

3.3.3.2. Positive Response

If the NETCONF server can satisfy the request, the server sends an OK element. For example:

```
<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 13: Successful delete subscription

3.3.3.3. Negative Response

If the NETCONF server cannot satisfy the request, the server sends an error-rpc element indicating the modification didn't work. For example:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-path
      xmlns:t="urn:ietf:params:xml:ns:netconf:notification:2.0">
      /t:identifier
    </error-path>
    <error-message xml:lang="en">
      no-such-subscription
    </error-message>
  </rpc-error>
</rpc-reply>
```

Figure 14: Unsuccessful delete subscription

3.4. Configured Subscriptions

Configured subscriptions are established, modified, and deleted using configuration operations against the top-level subtree of [subscribe] or [yang-push] via configuration interface. In this document, we focus on NETCONF operations. Any other configuration interface can be used to establish a configured subscription that uses NETCONF to push notifications to receivers. Configured subscriptions are supported by NETCONF servers using NETCONF Call Home [RFC8071]. Note that this document only covers configured subscriptions where the protocol of choice is NETCONF. In this section, we present examples of how to manage the configuration subscriptions using a NETCONF client. Key differences from dynamic subscriptions over NETCONF is that subscription lifetimes are decoupled from NETCONF sessions.

3.4.1. Establishing a Configured Subscription

For subscription establishment, a NETCONF client may send:

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
      <subscription>
        <identifier>
          1922
        </identifier>
        <stream>
          foo
        </stream>
        <receiver>
          <address>
            1.2.3.4
          </address>
          <port>
            1234
          </port>
          <protocol>
            netconf
          </protocol>
        </receiver>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>
```

Figure 15: Establish configured subscription

if the request is accepted, the server would reply:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 16: Response to a successful configuration subscription establishment

if the request is not accepted because the server cannot serve it, no configuration is changed. In this case the server may reply:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>resource-denied</error-tag>
    <error-severity>error</error-severity>
    <error-message xml:lang="en">
      Temporarily the server cannot serve this
      subscription due to the current workload.
    </error-message>
  </rpc-error>
</rpc-reply>
```

Figure 17: Response to a failed configured subscription establishment

For every configured receiver, once NETCONF transport session between the server and the receiver is recognized as active, the server will issue a "subscription-started" notification. After that, the server will send notifications to the receiver as per the subscription notification. The session is only intended for pushing notifications. Client request on that session SHOULD be ignored by the server.

The contents sent by the server on the Call Home session, once established, are identical to those in a dynamic subscription.

Note that the server assumes that the receiver is ready to accept notifications on the NETCONF session. This may require coordination between the client that configures the subscription and the clients for which the notifications are intended. This coordination is out of the scope of this document.

3.4.2. Call Home for Configured Subscriptions

Once this configuration is active, if NETCONF transport is needed but does not exist to one or more target IP address plus port, the server

initiates a transport session via [RFC8071] to those receiver(s) in the subscription using the address and port specified.

3.4.3. Full Establish Message Flow

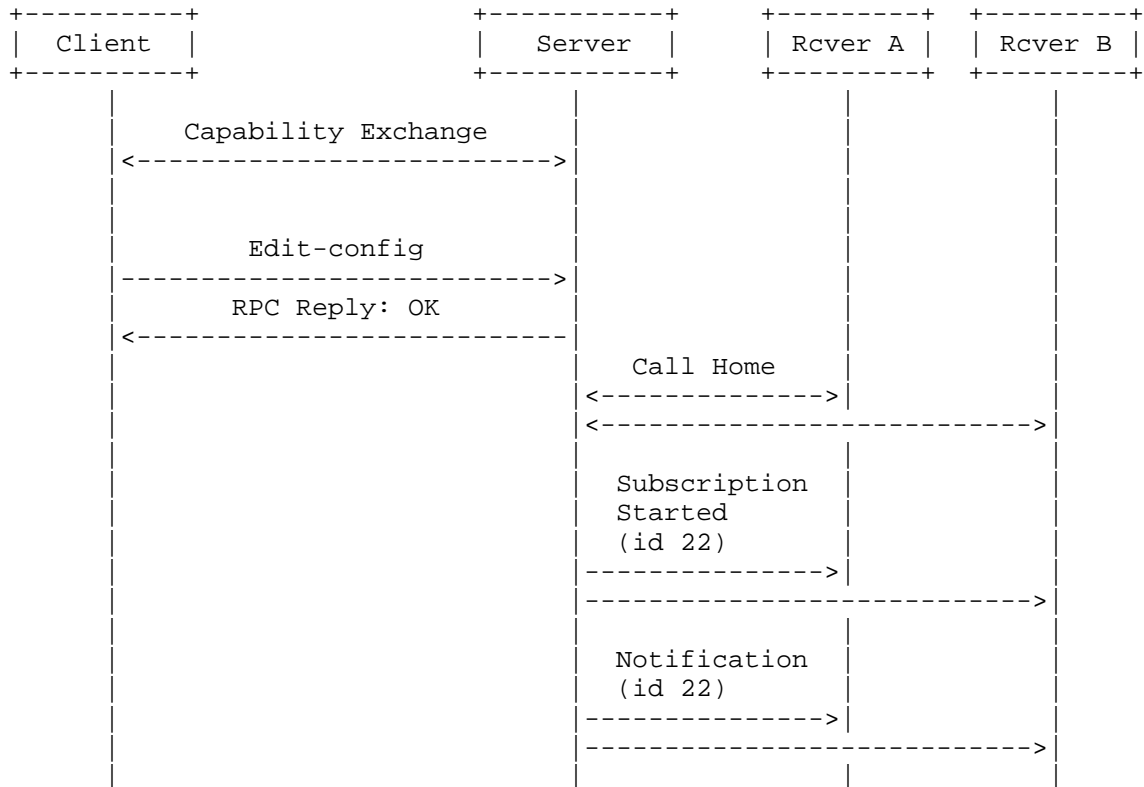


Figure 18: Message flow for configured subscription establishment

3.4.4. Modifying a Configured Subscription

Configured subscriptions can be modified using configuration operations against the top-level subtree subscription-config.

For example, the subscription established in the previous section could be modified as follows, choosing a different receiver:


```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
      <subscription>
        <identifier>
          1922
        </identifier>
        <stream>
          foo
        </stream>
        <receiver>
          <address>
            1.2.3.5
          </address>
          <port>
            1234
          </port>
        </receiver>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>
```

Figure 19: Modify configured subscription

if the request is accepted, the server would reply:

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 20: A successful configured subscription modification

3.4.4.1. Message Flow Example

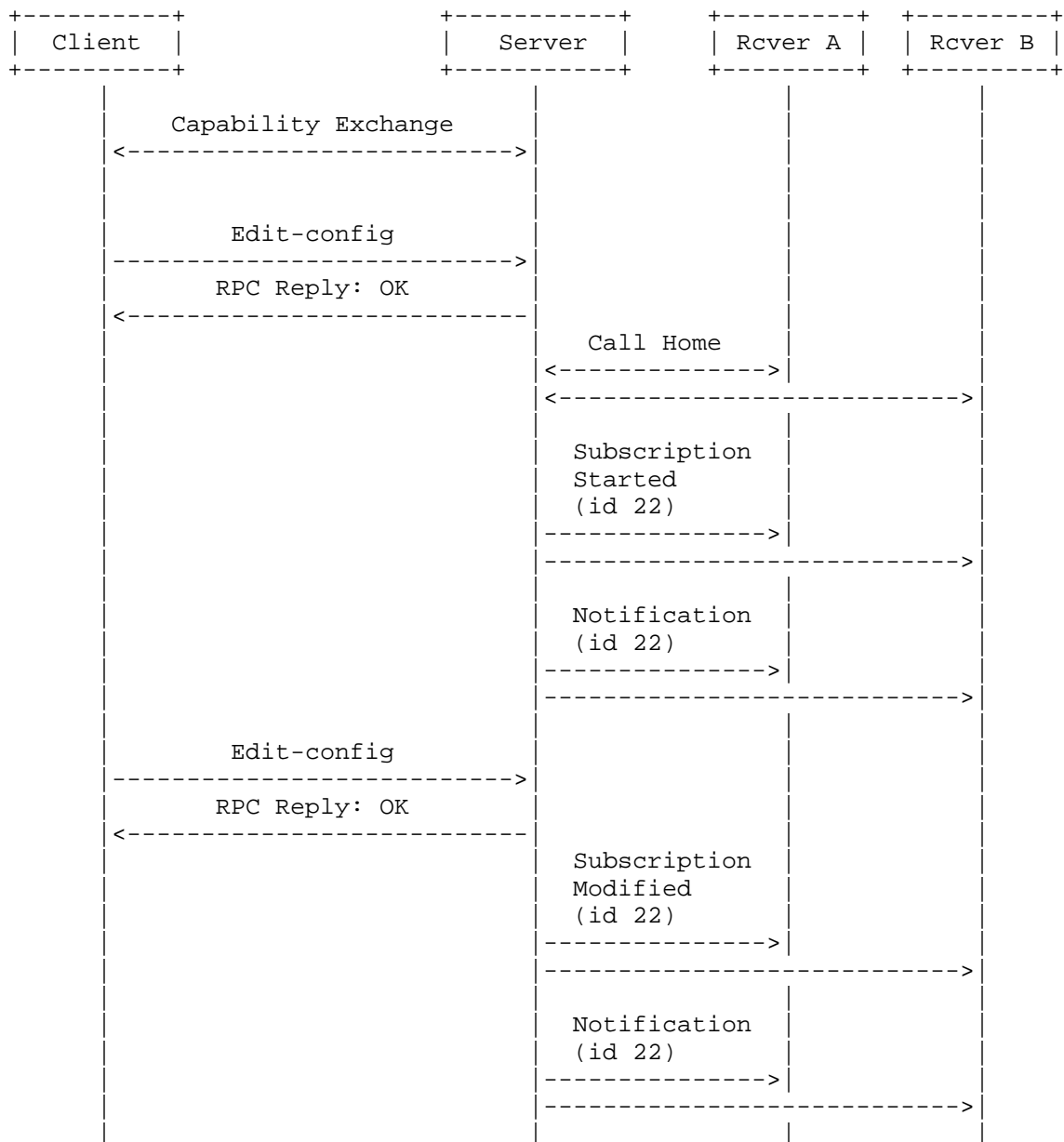


Figure 21: Message flow for subscription modification (configured subscription)

3.4.5. Deleting a Configured Subscription

Subscriptions can be deleted using configuration operations against the top-level subtree subscription-config. For example:

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns:xc="urn:ietf:params:xml:ns:netconf:notification:2.0">
      <subscription xc:operation="delete">
        <identifier>
          1922
        </identifier>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>

<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 22: Deleting a configured subscription

3.4.5.1. Message Flow Example

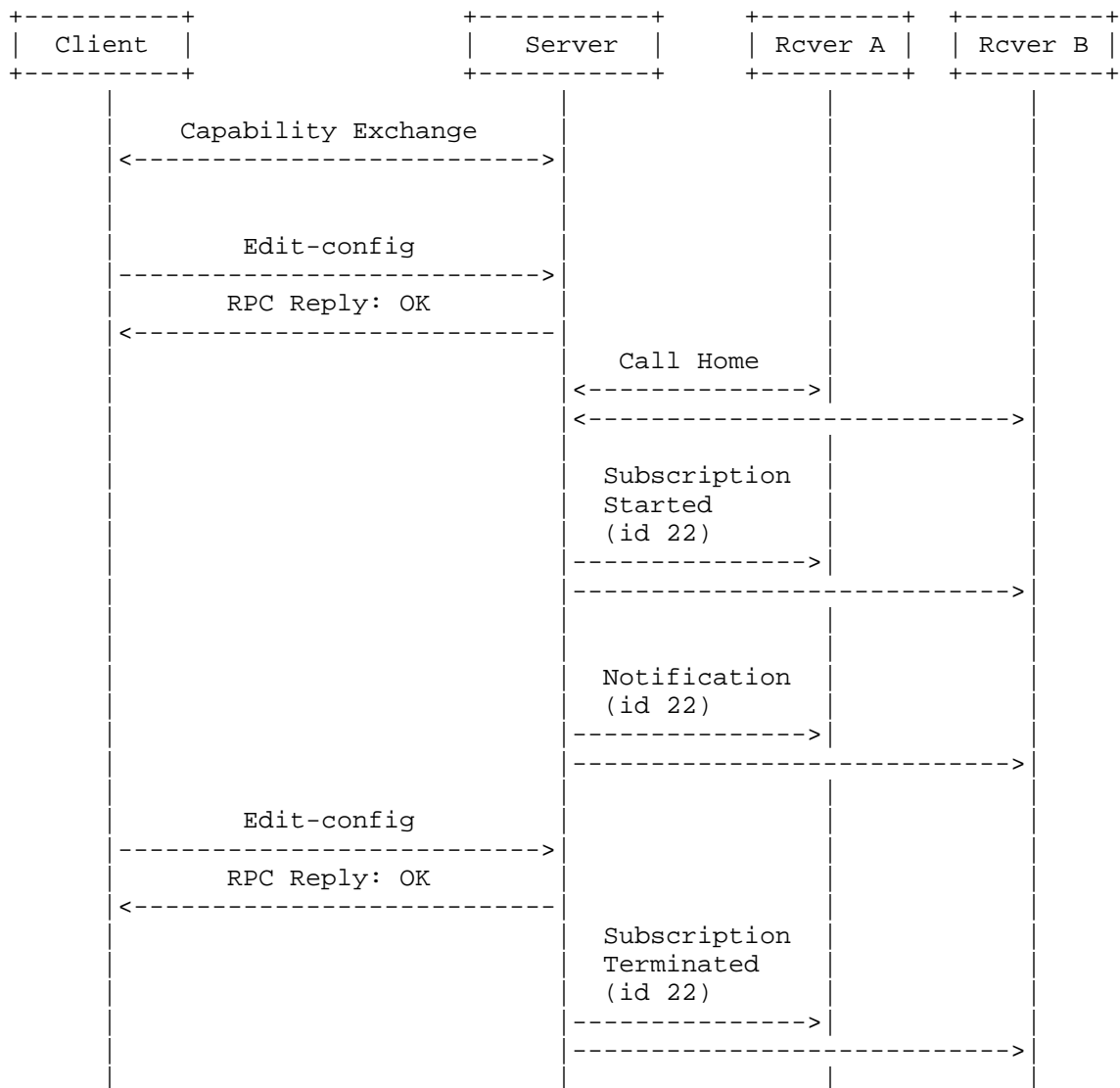


Figure 23: Message flow for subscription deletion (configured subscription)

3.4.6. Event (Data Plane) Notifications

Once a dynamic or configured subscription has been created, the NETCONF server sends (asynchronously) event notifications from the subscribed stream to receiver(s) over NETCONF. We refer to these as data plane notifications. The data model for Event Notifications is

defined in [subscribe]. This document extends that data model for supporting different encodings.

The following is an example of an event notification from [RFC6020]:

```
notification link-failure {
  description "A link failure has been detected";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf if-admin-status {
    type admin-status;
  }
  leaf if-oper-status {
    type oper-status;
  }
}
```

Figure 24: Definition of a event notification

This notification might result in the following, prior to it being placed into NETCONF. Note that the mandatory `eventTime` and `Subscription id` have been added.

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-id>39</subscription-id>
  <link-failure xmlns="http://acme.example.com/system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>
```

Figure 25: Event notification

In order to support JSON encoding, this document extends the data model for Event Notifications, adding a new element, i.e., `notification-content-json`. This element contains the event notification-specific tagged content in JSON. For the event notification above, the equivalent using JSON encoding would be

```

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-id>39</subscription-id>
  <notification-contents-json>
    {
      "acme-system:link-failure": {
        "if-name": "so-1/2/3.0",
        "if-admin-status": "up",
        "if-oper-status": "down"
      }
    }
  </notification-contents-json>
</notification>

```

Figure 26: Event notification using JSON encoding

3.4.7. Subscription State Notifications

In addition to data plane notifications, a publisher may send subscription state notifications (defined in [subscribe]) to indicate to receivers that an event related to the subscription management has occurred. Subscription state notifications cannot be filtered out. Next we exemplify them using both XML, and JSON encodings for the notification-specific content:

3.4.7.1. subscription-started and subscription-modified

A subscription-started would look like:

```

<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-started
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0"/>
    <identifier>39</identifier>
    <event-filter-type>xpath</event-filter-type>
    <event-filter xmlns:ex="http://example.com/event/1.0"
      select="/ex:event[ex:eventClass='fault' and
        (ex:severity='minor' or ex:severity='major'
        or ex:severity='critical')]">
    </subscription-started/>
  </notification>

```

Figure 27: subscription-started subscription state notification

The equivalent using JSON encoding would be:

```

<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-contents-json>
    {
      "notif-bis:subscription-started": {
        "identifier" : 39
      }
    }
  </notification-contents-json>
</notification>

```

Figure 28: subscription-started subscription state notification (JSON)

The subscription-modified is identical, with just the word "started" being replaced by "modified".

3.4.7.2. notification-complete, subscription-resumed, and replay-complete

A notification-complete would look like:

```

<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-complete
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <identifier>39</identifier>
  </notification-complete>
</notification>

```

Figure 29: notification-complete notification in XML

The equivalent using JSON encoding would be:

```

<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <notification-contents-json>
    {
      "netmod-notif:notification-complete": {
        "identifier" : 39
      }
    }
  </notification-contents-json>
</notification>

```

Figure 30: notification-complete notification in JSON

The subscription-resumed and replay-complete are virutally identical, with "notification-complete" simply being replaced by "subscription-resumed" and "replay-complete" in both encodings.

3.4.7.3. subscription-terminated and subscription-suspended

A subscription-terminated would look like:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-terminated
    xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
    <identifier>39</identifier>
    <error-id>no-such-subscription</error-id>
  </subscription-terminated>
</notification>
```

Figure 31: subscription-modified subscription state notification

The above, and the subscription-suspended are virutally identical, with "subscription-terminated" simply being replaced by "subscription-suspended".

3.4.7.4. Notification Message Flow Examples

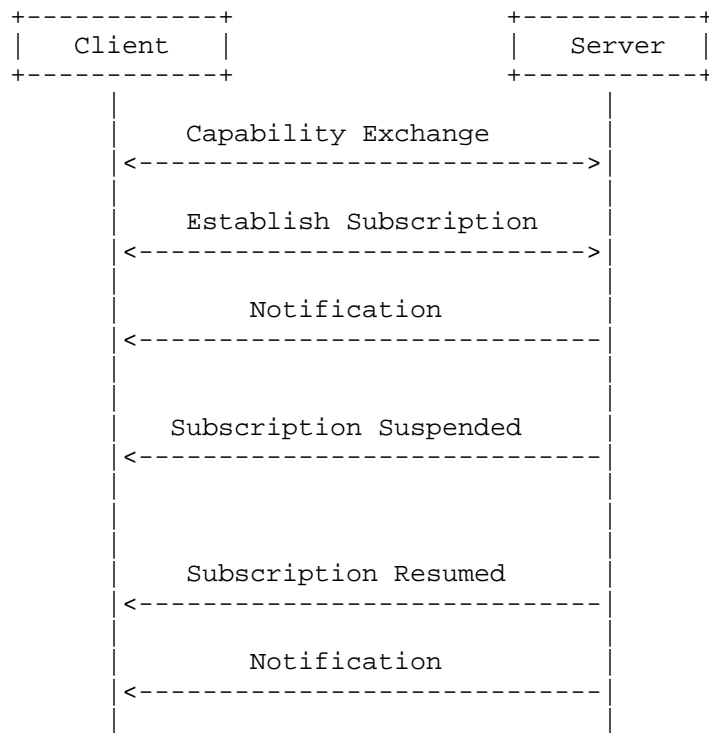


Figure 32: subscription-suspended and resumed notifications

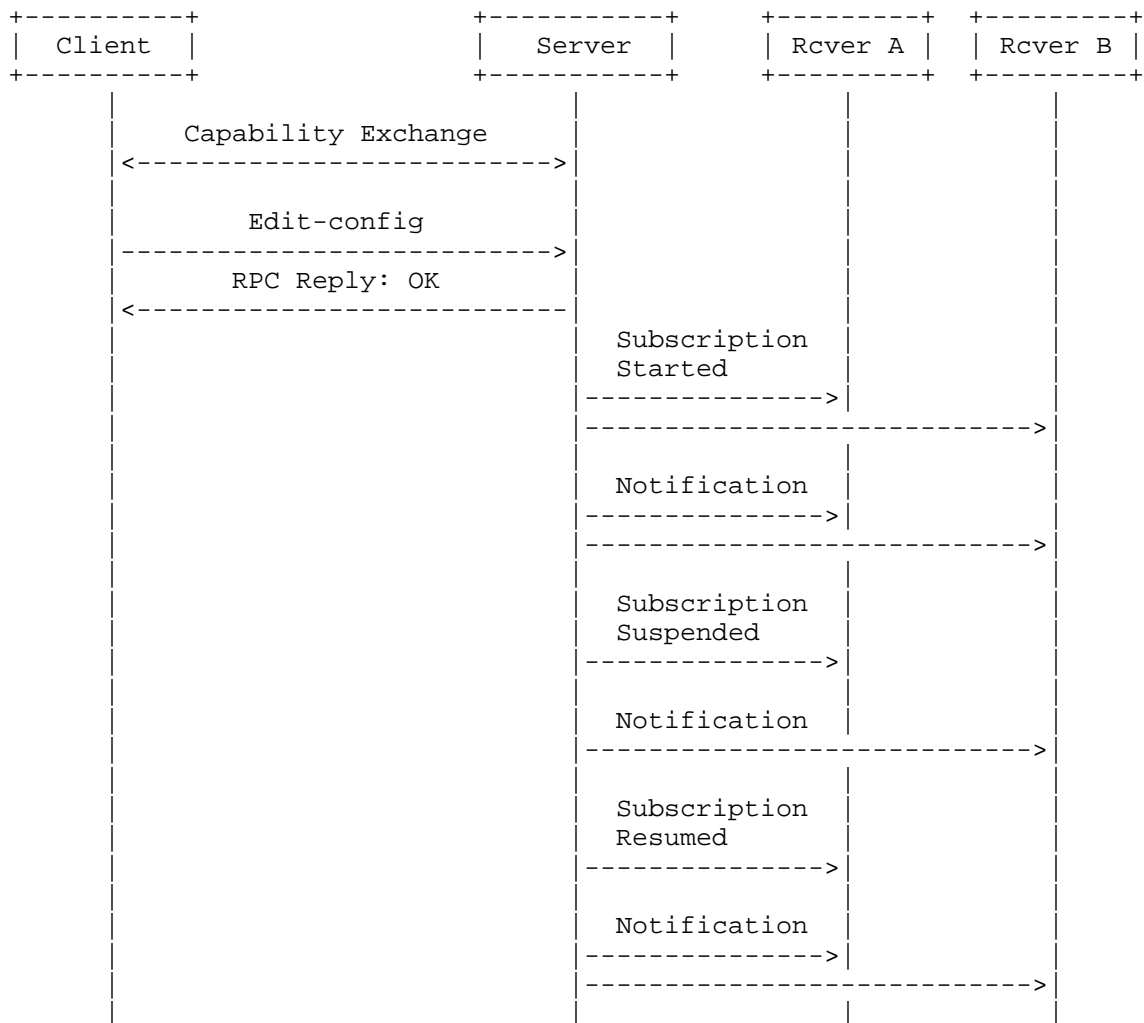


Figure 33: Suspended and resumed notifications for a single configured receiver

4. Interleave Capability

The `:interleave` capability is originally defined in [RFC5277]. It is incorporated in this document with essentially the same semantics. That is, the NETCONF server MUST receive, process, and respond to NETCONF requests on a session with active notification subscriptions. Note that subscription operations MUST be received, processed, and responded on a session with active notification subscriptions. This mandatory requirement together with the `:interleave` capability permits

a client performing all operations against a server using a single connection, allowing for better scalability with respect to the number of NETCONF sessions required to manage an entity. The :interleave capability is identified by the following string: urn:ietf:params:netconf:capability:interleave:1.0

5. Security Considerations

The <notification> elements are never sent before the transport layer and the NETCONF layer, including capabilities exchange, have been established and the manager has been identified and authenticated.

A secure transport must be used and the server must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of content involved.

The contents of notifications, as well as the names of event streams, may contain sensitive information and care should be taken to ensure that they are viewed only by authorized users. The NETCONF server MUST NOT include any content in a notification that the user is not authorized to view.

If a malicious or buggy NETCONF client sends a number of <establish-subscription>requests, then these subscriptions accumulate and may use up system resources. In such a situation, subscriptions MAY be terminated by terminating the suspect underlying NETCONF sessions. The server MAY also suspend or terminate a subset of the active subscriptions on the NETCONF session .

Configured subscriptions from one or more publishers could be used to overwhelm a receiver, which perhaps doesn't even support subscriptions. Clients that do not want pushed data need only terminate or refuse any transport sessions from the publisher.

The NETCONF Authorization Control Model [RFC6536] SHOULD be used to control and restrict authorization of subscription configuration. This control models permits specifying per-user permissions to receive specific event notification types. The permissions are specified as a set of access control rules.

6. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Andy Bierman, Yan Gang, Sharon Chisholm, Hector Trevino, Peipei Guo, Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, and Guangying Zheng.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<http://www.rfc-editor.org/info/rfc5277>>.
- [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>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <<http://www.rfc-editor.org/info/rfc8071>>.

7.2. Informative References

- [subscribe] Voit, Eric., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscribing to Event Notifications", April 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>>.
- [yang-push] Clemm, A., Gonzalez Prieto, A., Voit, Eric., Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", April 2017, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>>.

Appendix A. Open Items

(To be removed by RFC editor prior to publication)

- o Formal definition of: notification-contents-json. It depends on the formal definition of the notification element
- o Specify how to indicate a stream is not part of the NETCONF stream. It depends on draft-ietf-netconf-subscribed-notifications

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

B.1. v03 to v04

- o Added additional detail to "configured subscriptions"
- o Added interleave capability
- o Adjusted terminology to that in draft-ietf-netconf-subscribed-notifications
- o Corrected namespaces in examples

B.2. v01 to v03

- o Text simplifications throughout
- o v02 had no meaningful changes

B.3. v00 to v01

- o Added Call Home in solution for configured subscriptions.
- o Clarified support for multiple subscription on a single session. No need to support multiple create-subscription.
- o Added mapping between terminology in [yang-push] and [RFC6241] (the one followed in this document).
- o Editorial improvements.

Authors' Addresses

Alberto Gonzalez Prieto
VMware

Email: agonzalezprie@vmware.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

K. Watsen
Juniper Networks
J. Schoenwaelder
Jacobs University Bremen
July 3, 2017

RESTCONF Client and Server Models
draft-ietf-netconf-restconf-client-server-04

Abstract

This document defines two YANG modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server. Both modules support the TLS transport protocol with both standard RESTCONF and RESTCONF Call Home connections.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

This document contains references to other drafts in progress, both in the Normative References section, as well as in body text throughout. Please update the following references to reflect their final RFC assignments:

- o I-D.ietf-netconf-keystore
- o I-D.ietf-netconf-tls-client-server

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- o "XXXX" --> the assigned RFC value for this draft
- o "ZZZZ" --> the assigned RFC value for I-D.ietf-netconf-tls-client-server

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- o "2017-07-03" --> the publication date of this draft

The following Appendix section is to be removed prior to publication:

- o Appendix A. Change Log

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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 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	3
1.1. Terminology	3
1.2. Tree Diagrams	3
2. The RESTCONF Client Model	4
2.1. Tree Diagram	4
2.2. Example Usage	6
2.3. YANG Model	8
3. The RESTCONF Server Model	16
3.1. Tree Diagram	17
3.2. Example Usage	18
3.3. YANG Model	20
4. Security Considerations	29

5.	IANA Considerations	30
5.1.	The IETF XML Registry	30
5.2.	The YANG Module Names Registry	31
6.	Acknowledgements	31
7.	References	31
7.1.	Normative References	31
7.2.	Informative References	32
Appendix A.	Change Log	34
A.1.	server-model-09 to 00	34
A.2.	00 to 01	34
A.3.	01 to 02	34
A.4.	02 to 03	34
A.5.	03 to 04	34
Authors' Addresses	34

1. Introduction

This document defines two YANG [RFC7950] modules, one module to configure a RESTCONF client and the other module to configure a RESTCONF server [RFC8040]. Both modules support the TLS [RFC5246] transport protocol with both standard RESTCONF and RESTCONF Call Home connections [RFC8071].

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Tree Diagrams

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. The RESTCONF Client Model

EDITOR NOTE: Please ignore this section, it is incomplete.

The RESTCONF client model presented in this section supports both clients initiating connections to servers, as well as clients listening for connections from servers calling home.

This model supports the TLS transport protocol using the TLS client groupings defined in [I-D.ietf-netconf-tls-client-server].

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF client supports.

2.1. Tree Diagram

Just the container is displayed below, but there is also a grouping that the container is using.

Note: all lines are folded at column 71 with no '\ ' character.

```

module: ietf-restconf-client
  +--rw restconf-client
    +--rw initiate {initiate}?
      +--rw restconf-server* [name]
        +--rw name string
        +--rw (transport)
          +--:(tls) {tls-initiate}?
            +--rw tls
              +--rw endpoints
                +--rw endpoint* [name]
                  +--rw name string
                  +--rw address inet:host
                  +--rw port? inet:port-number
              +--rw server-auth
                +--rw trusted-ca-certs? leafref
                +--rw trusted-server-certs? leafref
              +--rw client-auth
                +--rw (auth-type)

```

```

|         |         +---:(certificate)
|         |         +---rw certificate?      leafref
+---rw hello-params
|         {tls-client-hello-params-config}?
|         +---rw tls-versions
|         |   +---rw tls-version*          identityref
+---rw cipher-suites
|         +---rw cipher-suite*             identityref
+---rw connection-type
|   +---rw (connection-type)?
|     +---:(persistent-connection)
|       +---rw persistent!
|       +---rw idle-timeout?              uint32
|       +---rw keep-alives
|       +---rw max-wait?                   uint16
|       +---rw max-attempts?              uint8
+---:(periodic-connection)
|   +---rw periodic!
|   +---rw idle-timeout?                  uint16
|   +---rw reconnect-timeout?            uint16
+---rw reconnect-strategy
|   +---rw start-with?                    enumeration
|   +---rw max-attempts?                  uint8
+---rw listen {listen}?
+---rw max-sessions?                      uint16
+---rw idle-timeout?                      uint16
+---rw endpoint* [name]
|   +---rw name                          string
+---rw (transport)
|   +---:(tls) {tls-listen}?
|     +---rw tls
|       +---rw address?                   inet:ip-address
|       +---rw port?                       inet:port-number
+---rw server-auth
|   +---rw trusted-ca-certs?               leafref
|   +---rw trusted-server-certs?           leafref
+---rw client-auth
|   +---rw (auth-type)
|   +---:(certificate)
|   +---rw certificate?                     leafref
+---rw hello-params
|   {tls-client-hello-params-config}?
|   +---rw tls-versions
|   |   +---rw tls-version*                identityref
+---rw cipher-suites
|   +---rw cipher-suite*                  identityref

```

2.2. Example Usage

The following example illustrates configuring a RESTCONF client to initiate connections, as well as listening for call-home connections.

This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

```
<restconf-client
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-client">

  <!-- RESTCONF servers to initiate connections to -->
  <initiate>
    <restconf-server>
      <name>corp-fw1</name>
      <tls>
        <endpoints>
          <endpoint>
            <name>corp-fw1.example.com</name>
            <address>corp-fw1.example.com</address>
          </endpoint>
          <endpoint>
            <name>corp-fw2.example.com</name>
            <address>corp-fw2.example.com</address>
          </endpoint>
        </endpoints>
        <server-auth>
          <trusted-server-certs>deployment-specific-ca-certs</trusted-server-cer
ts>
        </server-auth>
        <client-auth>
          <certificate>tls-ec-cert</certificate>
        </client-auth>
      </tls>
    </restconf-server>
  </initiate>

  <!-- endpoints to listen for RESTCONF Call Home connections on -->
  <listen>
    <endpoint>
      <name>Intranet-facing listener</name>
      <tls>
        <address>11.22.33.44</address>
        <server-auth>
          <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
          <trusted-server-certs>explicitly-trusted-server-certs</trusted-server-
certs>
        </server-auth>
        <client-auth>
          <certificate>tls-ec-cert</certificate>
        </client-auth>
      </tls>
    </endpoint>
  </listen>
</restconf-client>
```

2.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

```
<CODE BEGINS> file "ietf-restconf-client@2017-07-03.yang"

module ietf-restconf-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-client";
  prefix "rcc";

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-tls-client {
    prefix ts;
    revision-date 2017-06-13; // stable grouping definitions
    reference
      "RFC ZZZZ: TLS Client and Server Models";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/restconf/>
    WG List:  <mailto:restconf@ietf.org>

    Author:   Kent Watsen
              <mailto:kwatsen@juniper.net>

    Author:   Gary Wu
              <mailto:garywu@cisco.com>";

  description
    "This module contains a collection of YANG definitions for
    configuring RESTCONF clients.

    Copyright (c) 2014 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
```

without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

```
revision "2017-07-03" {
  description
    "Initial version";
  reference
    "RFC XXXX: RESTCONF Client and Server Models";
}

// Features

feature initiate {
  description
    "The 'initiate' feature indicates that the RESTCONF client
    supports initiating RESTCONF connections to RESTCONF servers
    using at least one transport (e.g., TLS, etc.).";
}

feature tls-initiate {
  description
    "The 'tls-initiate' feature indicates that the RESTCONF client
    supports initiating TLS connections to RESTCONF servers.";
  reference
    "RFC 8040: RESTCONF Protocol";
}

feature listen {
  description
    "The 'listen' feature indicates that the RESTCONF client
    supports opening a port to accept RESTCONF server call
    home connections using at least one transport (e.g.,
    TLS, etc.).";
}

feature tls-listen {
  description
    "The 'tls-listen' feature indicates that the RESTCONF client
    supports opening a port to listen for incoming RESTCONF
    server call-home TLS connections.";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}
```

```
}

container restconf-client {
  uses restconf-client;
  description
    "Top-level container for RESTCONF client configuration.";
}

grouping restconf-client {
  description
    "Top-level grouping for RESTCONF client configuration.";

  container initiate {
    if-feature initiate;
    description
      "Configures client initiating underlying TCP connections.";
    list restconf-server {
      key name;
      description
        "List of RESTCONF servers the RESTCONF client is to initiate
        connections to.";
      leaf name {
        type string;
        description
          "An arbitrary name for the RESTCONF server.";
      }
    }
    choice transport {
      mandatory true;
      description
        "Selects between available transports.";

      case tls {
        if-feature tls-initiate;
        container tls {
          description
            "Specifies TLS-specific transport configuration.";
          uses endpoints-container {
            refine endpoints/endpoint/port {
              default 443;
            }
          }
          uses ts:tls-client-grouping {
            refine "client-auth/auth-type" {
              mandatory true;
              description
                "RESTCONF clients MUST pass some authentication
                credentials.";
            }
          }
        }
      }
    }
  }
}
```



```
    }  
  }  
} // end tls  
  
} // end transport  
  
container connection-type {  
  description  
    "Indicates the kind of connection to use.";  
  choice connection-type {  
    description  
      "Selects between available connection types.";  
    case persistent-connection {  
      container persistent {  
        presence true;  
        description  
          "Maintain a persistent connection to the RESTCONF  
          server. If the connection goes down, immediately  
          start trying to reconnect to it, using the  
          reconnection strategy.  
  
          This connection type minimizes any RESTCONF server  
          to RESTCONF client data-transfer delay, albeit at  
          the expense of holding resources longer.";  
        leaf idle-timeout {  
          type uint32;  
          units "seconds";  
          default 86400; // one day;  
          description  
            "Specifies the maximum number of seconds that a  
            a RESTCONF session may remain idle. A RESTCONF  
            session will be dropped if it is idle for an  
            interval longer than this number of seconds.  
            If set to zero, then the client will never drop  
            a session because it is idle. Sessions that  
            have a notification subscription active are  
            never dropped.";  
        }  
      }  
      container keep-alives {  
        description  
          "Configures the keep-alive policy, to proactively  
          test the aliveness of the SSH/TLS server. An  
          unresponsive SSH/TLS server will be dropped after  
          approximately max-attempts * max-wait seconds.";  
        reference  
          "RFC 8071: NETCONF Call Home and RESTCONF Call  
          Home, Section 3.1, item S6";  
        leaf max-wait {
```

```
    type uint16 {
        range "1..max";
    }
    units seconds;
    default 30;
    description
        "Sets the amount of time in seconds after which
         if no data has been received from the SSH/TLS
         server, a SSH/TLS-level message will be sent
         to test the aliveness of the SSH/TLS server.";
}
leaf max-attempts {
    type uint8;
    default 3;
    description
        "Sets the maximum number of sequential keep-alive
         messages that can fail to obtain a response from
         the SSH/TLS server before assuming the SSH/TLS
         server is no longer alive.";
}
}
}
}
case periodic-connection {
    container periodic {
        presence true;
        description
            "Periodically connect to the RESTCONF server, so that
             the RESTCONF server may deliver messages pending for
             the RESTCONF client. The RESTCONF server must close
             the connection when it is ready to release it. Once
             the connection has been closed, the RESTCONF client
             will restart its timer until the next connection.";
        leaf idle-timeout {
            type uint16;
            units "seconds";
            default 300; // five minutes
            description
                "Specifies the maximum number of seconds that a
                 a RESTCONF session may remain idle. A RESTCONF
                 session will be dropped if it is idle for an
                 interval longer than this number of seconds.
                 If set to zero, then the server will never drop
                 a session because it is idle. Sessions that
                 have a notification subscription active are
                 never dropped.";
        }
        leaf reconnect-timeout {
```

```

    type uint16 {
        range "1..max";
    }
    units minutes;
    default 60;
    description
        "Sets the maximum amount of unconnected time the
         RESTCONF client will wait before re-establishing
         a connection to the RESTCONF server. The RESTCONF
         client may initiate a connection before this
         time if desired (e.g., to set configuration).";
}
}
}
}
container reconnect-strategy {
    description
        "The reconnection strategy directs how a RESTCONF client
         reconnects to a RESTCONF server, after discovering its
         connection to the server has dropped, even if due to a
         reboot. The RESTCONF client starts with the specified
         endpoint and tries to connect to it max-attempts times
         before trying the next endpoint in the list (round
         robin).";
    leaf start-with {
        type enumeration {
            enum first-listed {
                description
                    "Indicates that reconnections should start with
                     the first endpoint listed.";
            }
            enum last-connected {
                description
                    "Indicates that reconnections should start with
                     the endpoint last connected to. If no previous
                     connection has ever been established, then the
                     first endpoint configured is used. RESTCONF
                     clients SHOULD be able to remember the last
                     endpoint connected to across reboots.";
            }
        }
        default first-listed;
        description
            "Specifies which of the RESTCONF server's endpoints the
             RESTCONF client should start with when trying to connect
             to the RESTCONF server.";
    }
}

```

```
    leaf max-attempts {
      type uint8 {
        range "1..max";
      }
      default 3;
      description
        "Specifies the number times the RESTCONF client tries to
        connect to a specific endpoint before moving on to the
        next endpoint in the list (round robin).";
    }
  }
} // end restconf-server
} // end initiate

container listen {
  if-feature listen;
  description
    "Configures client accepting call-home TCP connections.";

  leaf max-sessions {
    type uint16;
    default 0;
    description
      "Specifies the maximum number of concurrent sessions
      that can be active at one time. The value 0 indicates
      that no artificial session limit should be used.";
  }

  leaf idle-timeout {
    type uint16;
    units "seconds";
    default 3600; // one hour
    description
      "Specifies the maximum number of seconds that a RESTCONF
      session may remain idle. A RESTCONF session will be dropped
      if it is idle for an interval longer than this number of
      seconds. If set to zero, then the server will never drop
      a session because it is idle. Sessions that have a
      notification subscription active are never dropped.";
  }

  list endpoint {
    key name;
    description
      "List of endpoints to listen for RESTCONF connections.";
    leaf name {
      type string;
      description

```

```

        "An arbitrary name for the RESTCONF listen endpoint.";
    }
    choice transport {
        mandatory true;
        description
            "Selects between available transports.";
        case tls {
            if-feature tls-listen;
            container tls {
                description
                    "TLS-specific listening configuration for inbound
                     connections.";
                leaf address {
                    type inet:ip-address;
                    description
                        "The IP address to listen for call-home connections.";
                }
                leaf port {
                    type inet:port-number;
                    default 4336;
                    description
                        "The port number to listen for call-home connections.";
                }
                uses ts:tls-client-grouping {
                    refine "client-auth/auth-type" {
                        mandatory true;
                        description
                            "RESTCONF clients MUST pass some authentication
                             credentials.";
                    }
                }
            }
        }
    } // end transport
} // end endpoint
} // end listen

} // end restconf-client

grouping endpoints-container {
    description
        "This grouping is used to configure a set of RESTCONF servers
         a RESTCONF client may initiate connections to.";
    container endpoints {
        description
            "Container for the list of endpoints.";
        list endpoint {

```

```
key name;
unique "address port";
min-elements 1;
ordered-by user;
description
    "A non-empty user-ordered list of endpoints for this RESTCONF
    client to try to connect to. Defining more than one enables
    high-availability.";
leaf name {
    type string;
    description
        "An arbitrary name for this endpoint.";
}
leaf address {
    type inet:host;
    mandatory true;
    description
        "The IP address or hostname of the endpoint. If a
        hostname is configured and the DNS resolution results
        in more than one IP address, the RESTCONF client
        will process the IP addresses as if they had been
        explicitly configured in place of the hostname.";
}
leaf port {
    type inet:port-number;
    description
        "The IP port for this endpoint. The RESTCONF client will
        use the IANA-assigned well-known port (set via a refine
        statement when uses) if no value is specified.";
}
}
}
```

<CODE ENDS>

3. The RESTCONF Server Model

The RESTCONF Server model presented in this section supports servers both listening for connections as well as initiating call-home connections.

This model supports the TLS transport protocol using the TLS server groupings defined in [I-D.ietf-netconf-tls-client-server].

All private keys and trusted certificates are held in the keystore model defined in [I-D.ietf-netconf-keystore].

YANG feature statements are used to enable implementations to advertise which parts of the model the RESTCONF server supports.

3.1. Tree Diagram

Just the container is displayed below, but there is also a grouping that the container is using.

Note: all lines are folded at column 71 with no '\' character.

```

module: ietf-restconf-server
  +--rw restconf-server
    +--rw listen {listen}?
      |   +--rw max-sessions?   uint16
      |   +--rw endpoint* [name]
      |   |   +--rw name      string
      |   |   +--rw (transport)
      |   |   |   +---:(tls) {tls-listen}?
      |   |   |   +--rw tls
      |   |   |   |   +--rw address?      inet:ip-address
      |   |   |   |   +--rw port?         inet:port-number
      |   |   |   |   +--rw certificates
      |   |   |   |   |   +--rw certificate* [name]
      |   |   |   |   |   |   +--rw name      leafref
      |   |   |   |   +--rw client-auth
      |   |   |   |   |   +--rw trusted-ca-certs?      leafref
      |   |   |   |   |   +--rw trusted-client-certs?  leafref
      |   |   |   |   +--rw cert-maps
      |   |   |   |   |   +--rw cert-to-name* [id]
      |   |   |   |   |   |   +--rw id          uint32
      |   |   |   |   |   |   +--rw fingerprint    x509c2n:tls-fingerprint
      |   |   |   |   |   |   +--rw map-type       identityref
      |   |   |   |   |   |   +--rw name          string
      |   |   |   +--rw hello-params
      |   |   |   |   {tls-server-hello-params-config}?
      |   |   |   +--rw tls-versions
      |   |   |   |   +--rw tls-version*  identityref
      |   |   |   +--rw cipher-suites
      |   |   |   |   +--rw cipher-suite*  identityref
      |   +--rw call-home {call-home}?
      |   |   +--rw restconf-client* [name]
      |   |   |   +--rw name              string
      |   |   |   +--rw (transport)
      |   |   |   |   +---:(tls) {tls-call-home}?
      |   |   |   |   +--rw tls
      |   |   |   |   |   +--rw endpoints
      |   |   |   |   |   |   +--rw endpoint* [name]
      |   |   |   |   |   |   |   +--rw name      string

```

```

|         |--rw address      inet:host
|         |--rw port?       inet:port-number
|--rw certificates
|   |--rw certificate* [name]
|   |--rw name             leafref
|--rw client-auth
|   |--rw trusted-ca-certs?   leafref
|   |--rw trusted-client-certs? leafref
|   |--rw cert-maps
|   |   |--rw cert-to-name* [id]
|   |   |--rw id              uint32
|   |   |--rw fingerprint    x509c2n:tls-fingerprint
|   |   |--rw map-type        identityref
|   |   |--rw name            string
|--rw hello-params
|   {tls-server-hello-params-config}?
|   |--rw tls-versions
|   |   |--rw tls-version*   identityref
|   |--rw cipher-suites
|   |   |--rw cipher-suite*  identityref
|--rw connection-type
|   |--rw (connection-type)?
|   |   +--:(persistent-connection)
|   |   |   |--rw persistent!
|   |   |   |--rw keep-alives
|   |   |   |   |--rw max-wait?      uint16
|   |   |   |   |--rw max-attempts?  uint8
|   |   +--:(periodic-connection)
|   |   |--rw periodic!
|   |   |--rw reconnect-timeout?    uint16
|--rw reconnect-strategy
|   |--rw start-with?      enumeration
|   |--rw max-attempts?    uint8

```

3.2. Example Usage

The following example illustrates configuring a RESTCONF server to listen for RESTCONF client connections, as well as configuring call-home to one RESTCONF client.

This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

```

<restconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server"
  xmlns:x509c2n="urn:ietf:params:xml:ns:yang:ietf-x509-cert-to-name">

  <!-- listening for TLS (HTTPS) connections -->

```



```

<listen>
  <endpoint>
    <name>netconf/tls</name>
    <tls>
      <address>11.22.33.44</address>
      <certificates>
        <certificate>
          <name>tls-ec-cert</name>
        </certificate>
      </certificates>
      <client-auth>
        <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
        <trusted-client-certs>explicitly-trusted-client-certs</trusted-client-
certs>
      <cert-maps>
        <cert-to-name>
          <id>1</id>
          <fingerprint>11:0A:05:11:00</fingerprint>
          <map-type>x509c2n:san-any</map-type>
        </cert-to-name>
        <cert-to-name>
          <id>2</id>
          <fingerprint>B3:4F:A1:8C:54</fingerprint>
          <map-type>x509c2n:specified</map-type>
          <name>scooby-doo</name>
        </cert-to-name>
      </cert-maps>
    </client-auth>
  </tls>
</endpoint>
</listen>

<!-- calling home to a RESTCONF client -->
<call-home>
  <restconf-client>
    <name>config-manager</name>
    <tls>
      <endpoints>
        <endpoint>
          <name>east-data-center</name>
          <address>22.33.44.55</address>
        </endpoint>
        <endpoint>
          <name>west-data-center</name>
          <address>33.44.55.66</address>
        </endpoint>
      </endpoints>
      <certificates>
        <certificate>

```

```

        <name>tls-ec-cert</name>
      </certificate>
    </certificates>
    <client-auth>
      <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
      <trusted-client-certs>explicitly-trusted-client-certs</trusted-client-
certs>
    <cert-maps>
      <cert-to-name>
        <id>1</id>
        <fingerprint>11:0A:05:11:00</fingerprint>
        <map-type>x509c2n:san-any</map-type>
      </cert-to-name>
      <cert-to-name>
        <id>2</id>
        <fingerprint>B3:4F:A1:8C:54</fingerprint>
        <map-type>x509c2n:specified</map-type>
        <name>scooby-doo</name>
      </cert-to-name>
    </cert-maps>
  </client-auth>
</tls>
<connection-type>
  <periodic>
    <idle-timeout>300</idle-timeout>
    <reconnect-timeout>60</reconnect-timeout>
  </periodic>
</connection-type>
<reconnect-strategy>
  <start-with>last-connected</start-with>
  <max-attempts>3</max-attempts>
</reconnect-strategy>
</restconf-client>
</call-home>

</restconf-server>

```

3.3. YANG Model

This YANG module imports YANG types from [RFC6991] and [RFC7407].

```

<CODE BEGINS> file "ietf-restconf-server@2017-07-03.yang"

module ietf-restconf-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-server";
  prefix "rcs";

```

```
//import ietf-netconf-acm {
//  prefix nacm;
//  reference
//    "RFC 6536: Network Configuration Protocol (NETCONF)
//    Access Control Model";
//}

import ietf-inet-types {
  prefix inet;
  reference
    "RFC 6991: Common YANG Data Types";
}

import ietf-x509-cert-to-name {
  prefix x509c2n;
  reference
    "RFC 7407: A YANG Data Model for SNMP Configuration";
}

import ietf-tls-server {
  prefix ts;
  revision-date 2017-06-13; // stable grouping definitions
  reference
    "RFC ZZZZ: TLS Client and Server Models";
}

organization
  "IETF NETCONF (Network Configuration) Working Group";

contact
  "WG Web:    <http://tools.ietf.org/wg/netconf/>
  WG List:    <mailto:netconf@ietf.org>

  WG Chair: Mehmet Ersue
              <mailto:mehmet.ersue@nsn.com>

  WG Chair: Mahesh Jethanandani
              <mailto:mjethanandani@gmail.com>

  Editor: Kent Watsen
              <mailto:kwatsen@juniper.net>";

description
  "This module contains a collection of YANG definitions for
  configuring RESTCONF servers.

  Copyright (c) 2014 IETF Trust and the persons identified as
```

authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

```
revision "2017-07-03" {
  description
    "Initial version";
  reference
    "RFC XXXX: RESTCONF Client and Server Models";
}

// Features

feature listen {
  description
    "The 'listen' feature indicates that the RESTCONF server
    supports opening a port to accept RESTCONF client connections
    using at least one transport (e.g., TLS, etc.).";
}

feature tls-listen {
  description
    "The 'tls-listen' feature indicates that the RESTCONF server
    supports opening a port to listen for incoming RESTCONF
    client connections.";
  reference
    "RFC XXXX: RESTCONF Protocol";
}

feature call-home {
  description
    "The 'call-home' feature indicates that the RESTCONF server
    supports initiating RESTCONF call home connections to REETCONF
    clients using at least one transport (e.g., TLS, etc.).";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature tls-call-home {
```

```
description
  "The 'tls-call-home' feature indicates that the RESTCONF server
  supports initiating connections to RESTCONF clients.";
reference
  "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
}

feature client-cert-auth {
  description
    "The client-cert-auth feature indicates that the RESTCONF
    server supports the ClientCertificate authentication scheme.";
  reference
    "RFC ZZZZ: Client Authentication over New TLS Connection";
}

// top-level container
container restconf-server {
  uses restconf-server;
  description
    "Top-level container for RESTCONF server configuration.";
}

grouping restconf-server {
  description
    "Top-level grouping for RESTCONF server configuration.";

  container listen {
    if-feature listen;
    description
      "Configures listen behavior";
    leaf max-sessions {
      type uint16;
      default 0; // should this be 'max'?
      description
        "Specifies the maximum number of concurrent sessions
        that can be active at one time. The value 0 indicates
        that no artificial session limit should be used.";
    }
  }
  list endpoint {
    key name;
    description
      "List of endpoints to listen for RESTCONF connections.";
    leaf name {
      type string;
      description
        "An arbitrary name for the RESTCONF listen endpoint.";
    }
  }
}
```

```
choice transport {
  mandatory true;
  description
    "Selects between available transports.";
  case tls {
    if-feature tls-listen;
    container tls {
      description
        "TLS-specific listening configuration for inbound
        connections.";
      leaf address {
        type inet:ip-address;
        description
          "The IP address of the interface to listen on. The
          TLS server will listen on all interfaces if no value
          is specified. Please note that some addresses have
          special meanings (e.g., '0.0.0.0' and ':::').";
      }
      leaf port {
        type inet:port-number;
        default 443;
        description
          "The local port number on this interface the TLS server
          listens on.";
      }
      uses ts:tls-server-grouping {
        refine "client-auth" {
          must 'trusted-ca-certs or trusted-client-certs';
          description
            "RESTCONF servers MUST be able to validate clients.";
        }
        augment "client-auth" {
          description
            "Augments in the cert-to-name structure.";
          uses cert-maps-grouping;
        }
      }
    } // end tls container
  } // end tls case
} // end transport
} // end endpoint
} // end listen

container call-home {
  if-feature call-home;
  description
    "Configures call-home behavior";
  list restconf-client {
```

```
key name;
description
  "List of RESTCONF clients the RESTCONF server is to
  initiate call-home connections to.";
leaf name {
  type string;
  description
    "An arbitrary name for the remote RESTCONF client.";
}
choice transport {
  mandatory true;
  description
    "Selects between TLS and any transports augmented in.";
  case tls {
    if-feature tls-call-home;
    container tls {
      description
        "Specifies TLS-specific call-home transport
        configuration.";
      uses endpoints-container {
        refine endpoints/endpoint/port {
          default 4336;
        }
      }
      uses ts:tls-server-grouping {
        refine "client-auth" {
          must 'trusted-ca-certs or trusted-client-certs';
          description
            "RESTCONF servers MUST be able to validate clients.";
        }
        augment "client-auth" {
          description
            "Augments in the cert-to-name structure.";
          uses cert-maps-grouping;
        }
      }
    }
  }
}
container connection-type {
  description
    "Indicates the RESTCONF client's preference for how the
    RESTCONF server's connection is maintained.";
  choice connection-type {
    description
      "Selects between available connection types.";
    case persistent-connection {
      container persistent {
```

```
presence true;
description
  "Maintain a persistent connection to the RESTCONF
  client. If the connection goes down, immediately
  start trying to reconnect to it, using the
  reconnection strategy.

  This connection type minimizes any RESTCONF client
  to RESTCONF server data-transfer delay, albeit at
  the expense of holding resources longer.";

container keep-alives {
  description
    "Configures the keep-alive policy, to proactively
    test the aliveness of the TLS client. An
    unresponsive TLS client will be dropped after
    approximately (max-attempts * max-wait)
    seconds.";
  reference
    "RFC 8071: NETCONF Call Home and RESTCONF Call
    Home, Section 3.1, item S6";
  leaf max-wait {
    type uint16 {
      range "1..max";
    }
    units seconds;
    default 30;
    description
      "Sets the amount of time in seconds after which
      if no data has been received from the TLS
      client, a TLS-level message will be sent to
      test the aliveness of the TLS client.";
  }
  leaf max-attempts {
    type uint8;
    default 3;
    description
      "Sets the maximum number of sequential keep-alive
      messages that can fail to obtain a response from
      the TLS client before assuming the TLS client is
      no longer alive.";
  }
}
}
}
case periodic-connection {
  container periodic {
    presence true;
```



```

        endpoint connected to across reboots.";
    }
}
default first-listed;
description
    "Specifies which of the RESTCONF client's endpoints the
    RESTCONF server should start with when trying to connect
    to the RESTCONF client.";
}
leaf max-attempts {
    type uint8 {
        range "1..max";
    }
    default 3;
    description
        "Specifies the number times the RESTCONF server tries to
        connect to a specific endpoint before moving on to the
        next endpoint in the list (round robin).";
}
}
}
}
}
}

```

```

grouping cert-maps-grouping {
    description
        "A grouping that defines a container around the
        cert-to-name structure defined in RFC 7407.";
    container cert-maps {
        uses x509c2n:cert-to-name;
        description
            "The cert-maps container is used by a TLS-based RESTCONF
            server to map the RESTCONF client's presented X.509
            certificate to a RESTCONF username. If no matching and
            valid cert-to-name list entry can be found, then the
            RESTCONF server MUST close the connection, and MUST NOT
            accept RESTCONF messages over it.";
        reference
            "RFC XXXX: The RESTCONF Protocol";
    }
}

```

```

grouping endpoints-container {
    description
        "This grouping is used by tls container for call-home
        configurations.";
}

```

```
container endpoints {
  description
    "Container for the list of endpoints.";
  list endpoint {
    key name;
    unique "address port";
    min-elements 1;
    ordered-by user;
    description
      "User-ordered list of endpoints for this RESTCONF client.
      Defining more than one enables high-availability.";
    leaf name {
      type string;
      description
        "An arbitrary name for this endpoint.";
    }
    leaf address {
      type inet:host;
      mandatory true;
      description
        "The IP address or hostname of the endpoint.  If a
        hostname is configured and the DNS resolution results
        in more than one IP address, the RESTCONF server
        will process the IP addresses as if they had been
        explicitly configured in place of the hostname.";
    }
    leaf port {
      type inet:port-number;
      description
        "The IP port for this endpoint.  The RESTCONF server will
        use the IANA-assigned well-known port if no value is
        specified.";
    }
  }
}
```

<CODE ENDS>

4. Security Considerations

The YANG module defined in this document uses a grouping defined in [I-D.ietf-netconf-tls-client-server]. Please see the Security

Considerations section in that document for concerns related that grouping.

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

5. IANA Considerations

5.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-restconf-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-restconf-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the the following registrations are requested:

name:	ietf-restconf-client
namespace:	urn:ietf:params:xml:ns:yang:ietf-restconf-client
prefix:	ncc
reference:	RFC XXXX
name:	ietf-restconf-server
namespace:	urn:ietf:params:xml:ns:yang:ietf-restconf-server
prefix:	ncs
reference:	RFC XXXX

6. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, Balazs Kovacs, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Phil Shafer, Sean Turner, and Bert Wijnen.

Juergen Schoenwaelder and was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

7. References

7.1. Normative References

- [I-D.ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-02 (work in progress), June 2017.
- [I-D.ietf-netconf-tls-client-server]
Watsen, K. and G. Wu, "TLS Client and Server Models", draft-ietf-netconf-tls-client-server-03 (work in progress), June 2017.

- [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>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, DOI 10.17487/RFC7407, December 2014, <<http://www.rfc-editor.org/info/rfc7407>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.

7.2. Informative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [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>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home",
RFC 8071, DOI 10.17487/RFC8071, February 2017,
<<http://www.rfc-editor.org/info/rfc8071>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Added in new features 'listen' and 'call-home' so future transports can be augmented in.

A.2. 00 to 01

- o Renamed "keychain" to "keystore".

A.3. 01 to 02

- o Filled in previously missing 'ietf-restconf-client' module.
- o Updated the ietf-restconf-server module to accomodate new grouping 'ietf-tls-server-grouping'.

A.4. 02 to 03

- o Refined use of tls-client-grouping to add a must statement indicating that the TLS client must specify a client-certificate.
- o Changed restconf-client??? to be a grouping (not a container).

A.5. 03 to 04

- o Added RFC 8174 to Requirements Language Section.
- o Replaced refine statement in ietf-restconf-client to add a mandatory true.
- o Added refine statement in ietf-restconf-server to add a must statement.
- o Now there are containers and groupings, for both the client and server models.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

E. Voit
A. Gonzalez Prieto
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Clemm
Huawei
A. Bierman
YumaWorks
March 13, 2017

Restconf and HTTP Transport for Event Notifications
draft-ietf-netconf-restconf-notif-02

Abstract

This document defines Restconf, HTTP2, and HTTP1.1 bindings for the transport of Subscription requests and corresponding push updates. Being subscribed may be either Event Notifications or objects or subtrees of YANG Datastores.

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 September 14, 2017.

Copyright Notice

Copyright (c) 2017 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	3
3. Solution	3
3.1. Dynamic YANG Subscription with RESTCONF control	3
3.2. Subscription Multiplexing	6
4. Encoded Subscription and Event Notification Examples	7
4.1. Restconf Subscription and Events over HTTP1.1	7
4.2. Event Notification over HTTP2	12
5. Security Considerations	12
6. Acknowledgments	13
7. References	13
7.1. Normative References	13
7.2. Informative References	14
Appendix A. End-to-End Deployment Guidance	14
A.1. Call Home	14
A.2. TLS Heartbeat	15
Appendix B. Issues being worked and resolved	15
B.1. Unresolved Issues	15
Appendix C. Changes between revisions	15
Authors' Addresses	16

1. Introduction

Mechanisms to support Event subscription and push are defined in [sn]. Enhancements to [sn] which enable YANG Datastore subscription and push are defined in [yang-push]. This document provides a transport specification for these protocols over Restconf and HTTP. Driving these requirements is [RFC7923].

The streaming of Subscription Event Notifications has synergies with HTTP2 streams. Benefits which can be realized when transporting events directly HTTP2 [RFC7540] include:

- o Elimination of head-of-line blocking
- o Weighting and proportional dequeuing of Events from different subscriptions
- o Explicit precedence in subscriptions so that events from one subscription must be sent before another dequeues

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 use the definitions from [sn]: Configured Subscription, Dynamic Subscription, Event Notification, Publisher, Receiver, Subscriber, Subscription.

3. Solution

Event subscription is defined in [sn], YANG Datastore subscription is defined in [yang-push]. This section specifies transport mechanisms applicable to both.

3.1. Dynamic YANG Subscription with RESTCONF control

Dynamic Subscriptions for both [sn] and its [yang-push] augmentations are configured and managed via signaling messages transported over [RFC8040]. These interactions will be accomplished via a Restconf POST into RPCs located on the Publisher. HTTP responses codes will indicate the results of the interaction with the Publisher. An HTTP status code of 200 is the proper response to a successful <establish-subscription> RPC call. The successful <establish-subscription> will result in a HTTP message with returned subscription URI on a logically separate mechanism than was used for the original Restconf POST. This mechanism is via a parallel TCP connection in the case of HTTP 1.x, or in the case of HTTP2 via a separate HTTP stream within the HTTP connection. When a being returned by the Publisher, failure will be indicated by error codes transported in payload.

Once established, the resulting stream of Event Notifications are then delivered via SSE for HTTP1.1 and via HTTP Data for HTTP2.

3.1.1. Call Flow for HTTP2

Requests to [yang-push] augmented RPCs are sent on one or more HTTP2 streams indicated by (a) in Figure 2. Event Notifications related to a single subscription are pushed on a unique logical channel (b). In the case below, a newly established subscription has its events pushed over HTTP2 stream (7).

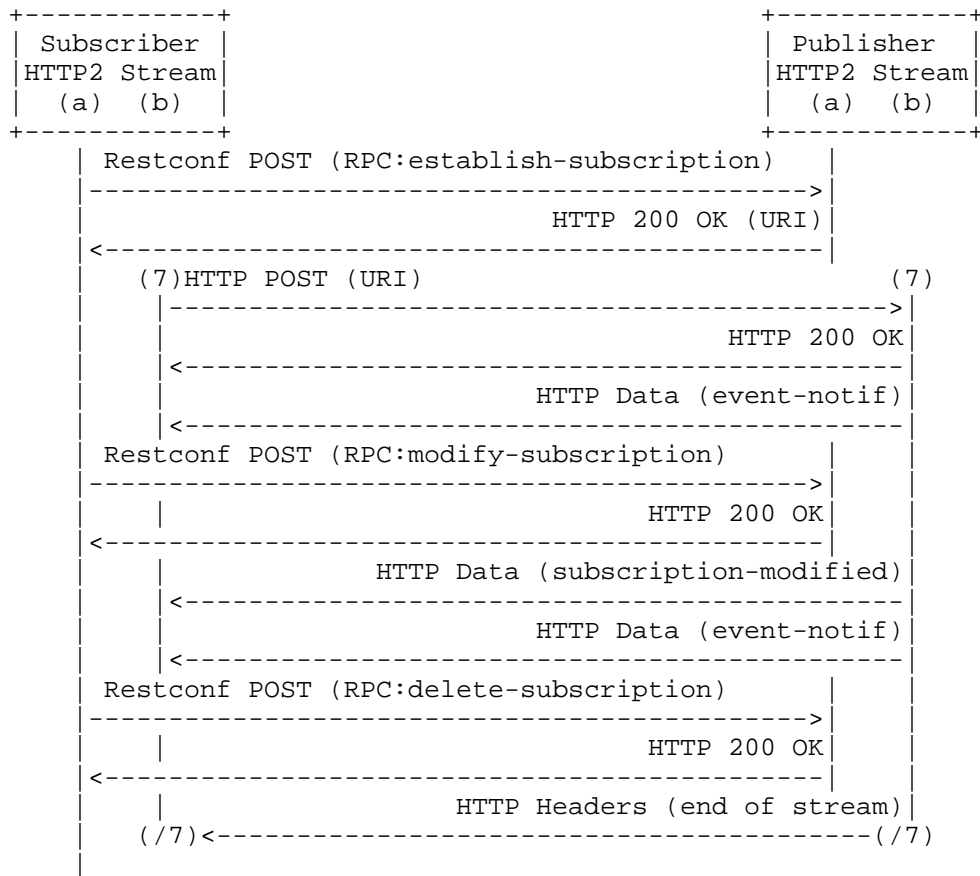


Figure 1: Dynamic with HTTP2

3.1.2. Call flow for HTTP1.1

Requests to [yang-push] RPCs are sent on the TCP connection indicated by (a). Event Notifications are pushed on a separate connection (b). This connection (b) will be used for all Event Notifications across all subscriptions.

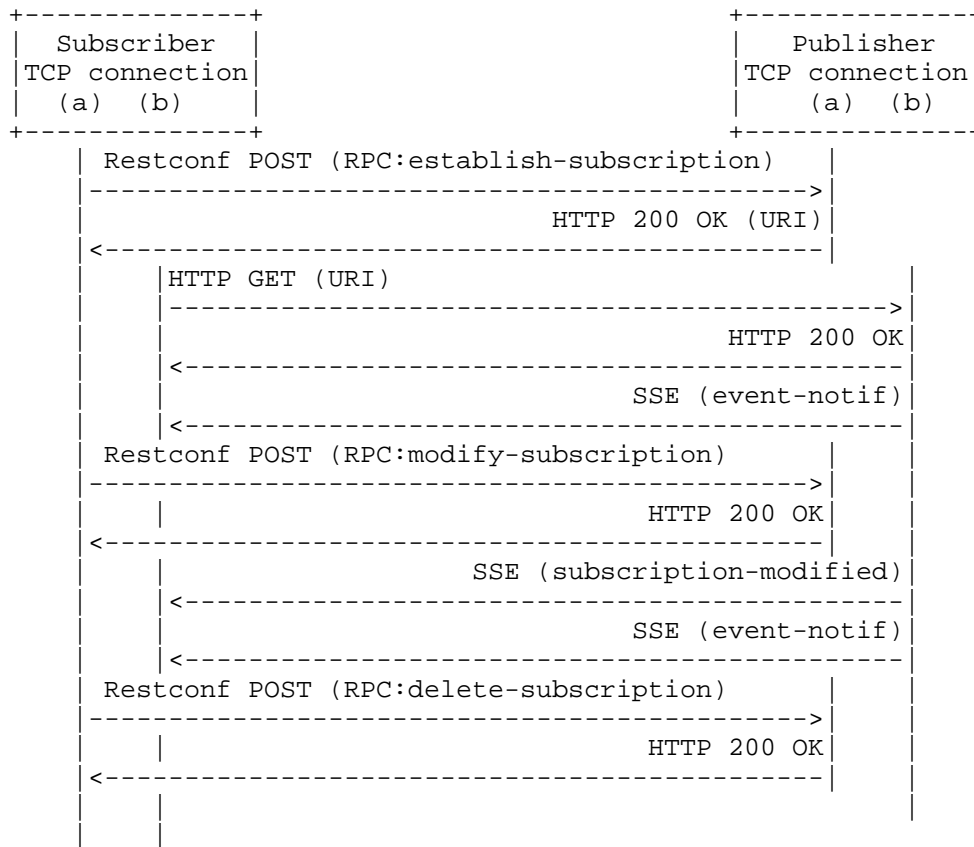


Figure 2: Dynamic with HTTP1.1

3.1.3. Configured Subscription over HTTP2

With a Configured Subscription, all information needed to establish a secure relationship with that Receiver is available on the Publisher. With this information, the Publisher will establish a secure transport connection with the Receiver and then begin pushing the Event Notifications to the Receiver. Since Restconf might not exist on the Receiver, it is not desirable to require that such Event Notifications be pushed with any dependency on Restconf. Nor is there value which Restconf provides on top of HTTP. Therefore in place of Restconf, a TLS secured HTTP2 Client connection must be established with an HTTP2 Server located on the Receiver. Event Notifications will then be sent as part of an extended HTTP POST to the Receiver.

POST messages will be addressed to HTTP augmentation code on the Receiver capable of accepting and responding to Event Notifications. The first POST message must be a subscription-started notification. Push update notifications must not be sent until the receipt of an HTTP 200 OK for this initial notification. The 200 OK will indicate that the Receiver is ready for Event Notifications. At this point a Subscription must be allocated its own HTTP2 stream. Figure 4 depicts this message flow.

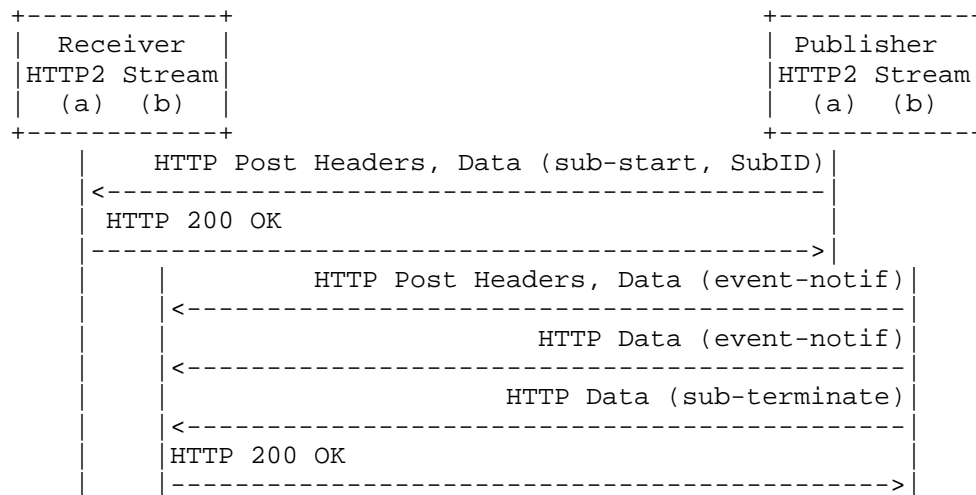


Figure 3: Configured over HTTP2

As the HTTP2 transport is available to the Receiver, the Publisher should:

- o take any subscription-priority and copy it into the HTTP2 stream priority, and
- o take a subscription-dependency if it has been provided and map the HTTP2 stream for the parent subscription into the HTTP2 stream dependency.

3.2. Subscription Multiplexing

It is possible that updates might be delivered in a different sequence than generated. Reasons for this might include (but are not limited to):

- o replay of pushed updates

- o temporary loss of transport connectivity, with update buffering and different dequeuing priorities per Subscription
- o population, marshalling and bundling of independent Subscription Updates, and

Therefore each Event Notification will include a timestamp to ensure that a Receiver understands the time when a that update was generated. Use of this timestamp can give an indication of the state of objects at a Publisher when state-entangled information is received across different subscriptions. The use of the latest Event Notification timestamp for a particular object update can introduce errors. So when state-entangled updates have inconsistent object values and temporally close timestamps, a Receiver might consider performing a GET to validate the current state of a Publisher.

4. Encoded Subscription and Event Notification Examples

Transported updates will contain context data for one or more Event Notifications. Each transported Event Notification will contain several parameters:

4.1. Restconf Subscription and Events over HTTP1.1

Subscribers can dynamically learn whether a RESTCONF server supports various types of Event or Yang datastore subscription capabilities. This is done by issuing an HTTP request OPTIONS, HEAD, or GET on the stream. Some examples building upon the Call flow for HTTP1.1 from Section 3.2.2 are:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/  
    streams/stream=yang-push HTTP/1.1  
Host: example.com  
Accept: application/yang.data+xml
```

If the server supports it, it may respond


```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
<stream xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
  <name>yang-push</name>
  <description>Yang push stream</description>
  <access>
    <encoding>xml</encoding>
    <location>https://example.com/streams/yang-push-xml
  </access>
  <access>
    <encoding>json</encoding>
    <location>https://example.com/streams/yang-push-json
  </access>
</stream>
```

If the server does not support any form of subscription, it may respond

```
HTTP/1.1 404 Not Found
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
```

Subscribers can determine the URL to receive updates by sending an HTTP GET as a request for the "location" leaf with the stream list entry. The stream to use for may be selected from the Event Stream list provided in the capabilities exchange. Note that different encodings are supporting using different Event Stream locations. For example, the Subscriber might send the following request:

```
GET /restconf/data/ietf-restconf-monitoring:restconf-state/
    streams/stream=yang-push/access=xml/location HTTP/1.1
Host: example.com
Accept: application/yang.data+xml
```

The Publisher might send the following response:

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
  <location
    xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-monitoring">
    https://example.com/streams/yang-push-xml
  </location>
```

To subscribe and start receiving updates, the subscriber can then send an HTTP GET request for the URL returned by the Publisher in the request above. The accept header must be "text/event-stream". The

Publisher uses the Server Sent Events [W3C-20150203] transport strategy to push filtered Event Notifications from the Event stream.

The Publisher MUST support individual parameters within the POST request body for all the parameters of a subscription. The only exception is the encoding, which is embedded in the URI. An example of this is:

```
// subtree filter = /foo
// periodic updates, every 5 seconds
POST /restconf/operations/ietf-event-notifications:
  establish-subscription HTTP/1.1
  Host: example.com
  Content-Type: application/yang-data+json

  {
    "ietf-event-notifications:input" : {
      ?stream?: ?push-data"
      ?period" : 5,
      "xpath-filter" : ?/ex:foo[starts-with(?bar?.?some']"
    }
  }
```

Should the publisher not support the requested subscription, it may reply:

```

HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+xml
  <errors xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
    <error>
      <error-type>application</error-type>
      <error-tag>operation-not-supported</error-tag>
      <error-severity>error</error-severity>
      <error-message>Xpath filters not supported</error-message>
      <error-info>
        <supported-subscription xmlns="urn:ietf:params:xml:ns:
          netconf:datastore-push:1.0">
          <subtree-filter/>
        </supported-subscription>
      </error-info>
    </error>
  </errors>

```

with an equivalent JSON encoding representation of:

```

HTTP/1.1 501 Not Implemented
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.errors+json
{
  "ietf-restconf:errors": {
    "error": {
      "error-type": "protocol",
      "error-tag": "operation-not-supported",
      "error-message": "Xpath filters not supported."
      "error-info": {
        "datastore-push:supported-subscription": {
          "subtree-filter": [null]
        }
      }
    }
  }
}

```

The following is an example of a pushed Event Notification data for the Subscription above. It contains a subtree with root foo that contains a leaf called bar:

XML encoding representation:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <subscription-id xmlns="urn:ietf:params:xml:ns:restconf:
    datastore-push:1.0">
    my-sub
  </subscription-id>
  <eventTime>2015-03-09T19:14:56.233Z</eventTime>
  <datastore-contents xmlns="urn:ietf:params:xml:ns:restconf:
    datastore-push:1.0">
    <foo xmlns="http://example.com/yang-push/1.0">
      <bar>some_string</bar>
    </foo>
  </datastore-contents>
</notification>
```

Or with the equivalent YANG over JSON encoding representation as defined in [RFC7951]:

```
{
  "ietf-restconf:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56.233Z",
    "datastore-push:datastore-contents": {
      "example-mod:foo": { "bar": "some_string" }
    }
  }
}
```

To modify a Subscription, the subscriber issues another POST request on the provided URI using the same subscription-id as in the original request. For example, to modify the update period to 10 seconds, the subscriber may send:

```
POST /restconf/operations/ietf-event-notifications:
  modify-subscription HTTP/1.1
  Host: example.com
  Content-Type: application/yang-data+json

  {
    "ietf-event-notifications:input" : {
      ?subscription-id?: 100,
      ?period" : 10
    }
  }
```

To delete a Subscription, the Subscriber issues a DELETE request on the provided URI using the same subscription-id as in the original request

4.2. Event Notification over HTTP2

The basic encoding will look as below. It will consists of a JSON representation wrapped in an HTTP2 header.

HyperText Transfer Protocol 2

Stream: HEADERS, Stream ID: 5

Header: :method: POST

Stream: HEADERS, Stream ID: 5

```
{
  "ietf-yangpush:notification": {
    "datastore-push:subscription-id": "my-sub",
    "eventTime": "2015-03-09T19:14:56.233Z",
    "datastore-push:datastore-contents": {
      "foo": { "bar": "some_string" }
    }
  }
}
```

5. Security Considerations

Subscriptions could be used to intentionally or accidentally overload the resources of a Publisher. For this reason, it is important that the Publisher has the ability to prioritize the establishment and push of Event Notifications where there might be resource exhaust potential. In addition, a server needs to be able to suspend existing Subscriptions when needed. When this occurs, the subscription status must be updated accordingly and the Receivers notified.

A Subscription could be used to attempt retrieve information for which a Receiver has no authorized access. Therefore it is important that data pushed via a Subscription is authorized equivalently with regular data retrieval operations. Data being pushed to a Receiver needs therefore to be filtered accordingly, just like if the data were being retrieved on-demand. The Netconf Authorization Control Model [RFC6536] applies even though the transport is not NETCONF.

One or more Publishers of Configured Subscriptions could be used to overwhelm a Receiver which doesn't even support Subscriptions. There are two protections here. First Event Notifications for Configured Subscriptions MUST only be transmittable over Encrypted transports. Clients which do not want pushed Event Notifications need only

terminate or refuse any transport sessions from the Publisher. Second, the HTTP transport augmentation on the Receiver must send an HTTP 200 OK to a subscription started notification before the Publisher starts streaming any events.

One or more Publishers could overwhelm a Receiver which is unable to control or handle the volume of Event Notifications received. In deployments where this might be a concern, HTTP2 transport such as HTTP2) should be selected.

6. Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from: Susan Hares, Tim Jenkins, Balazs Lengyel, Kent Watsen, Michael Scharf, and Guangying Zheng.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <<http://www.rfc-editor.org/info/rfc6520>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

[sn] Voit, E., Clemm, A., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to Event Notifications", February 2017, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>>.

7.2. Informative References

- [RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <<http://www.rfc-editor.org/info/rfc7923>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.
- [RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <<http://www.rfc-editor.org/info/rfc8071>>.
- [W3C-20150203] "Server-Sent Events, World Wide Web Consortium CR CR-eventsource-20121211", February 2015, <<https://www.w3.org/TR/2015/REC-eventsource-20150203/>>.
- [yang-push] Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", March 2017, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>>.

Appendix A. End-to-End Deployment Guidance

Several technologies are expected to be seen within a deployment to achieve security and ease-of-use requirements. These are not necessary for an implementation of this specification, but will be useful to consider when considering the operational context.

A.1. Call Home

Pub/Sub implementations should have the ability to transparently incorporate 'call home' [RFC8071] so that secure TLS connections can originate from the desired device.

A.2. TLS Heartbeat

HTTP sessions might not quickly allow a Subscriber to recognize when the communication path has been lost from the Publisher. To recognize this, it is possible for a Receiver to establish a TLS heartbeat [RFC6520]. In the case where a TLS heartbeat is included, it should be sent just from Receiver to Publisher. Loss of the heartbeat should result in any Subscription related TCP sessions between those endpoints being torn down. The subscription can then attempt to re-establish.

Appendix B. Issues being worked and resolved

(To be removed by RFC editor prior to publication)

B.1. Unresolved Issues

GRPC compatibility 1: Mechanisms for HTTP2 to GRPC mapping need to be considered. There is a good start there as this draft only uses POST, not GET. As GET is used in RESTCONF for capabilities discovery, we have some backwards compatibility issues with existing IETF drafts. Possible options to address are (1) provide a POST method for anything done by GET in RESTCONF, (2) await support of GET by GRPC, or (3) tunnel RESTCONF's GET messages within a GRPC POST.

GRPC compatibility 2: We need to expose a method against which POST is done as events begin on a stream. See Stream 7 in figure 2. Can only send traffic to a method, not a URI. URI points to a method, not a resource.

Need to add into document examples of Event streams. Document only includes yang-push examples at this point.

We need to reference the viable encodings of notifications.

Appendix C. Changes between revisions

(To be removed by RFC editor prior to publication)

v01 - v02

- o Removed sections now redundant with [sn] and [yang-push] such as: mechanisms for subscription maintenance, terminology definitions, stream discovery.
- o 3rd party subscriptions are out-of-scope.
- o SSE only used with Restconf and HTTP1.1 Dynamic Subscriptions

- o Timeframes for event tagging are self-defined.
- o Clean-up of wording, references to terminology, section numbers.

v00 - v01

- o Removed the ability for more than one subscription to go to a single HTTP2 stream.
- o Updated call flows. Extensively.
- o SSE only used with Restconf and HTTP1.1 Dynamic Subscriptions
- o HTTP is not used to determine that a Receiver has gone silent and is not Receiving Event Notifications
- o Many clean-ups of wording and terminology

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alberto Gonzalez Prieto
Cisco Systems

Email: albertgo@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 15, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Systems
June 13, 2017

SSH Client and Server Models
draft-ietf-netconf-ssh-client-server-03

Abstract

This document defines three YANG modules: the first defines groupings for a generic SSH client, the second defines groupings for a generic SSH server, and the third defines common identities and groupings used by both the client and the server. It is intended that these groupings will be used by applications using the SSH protocol.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

This document contains references to other drafts in progress, both in the Normative References section, as well as in body text throughout. Please update the following references to reflect their final RFC assignments:

- o I-D.ietf-netconf-keystore

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- o "XXXX" --> the assigned RFC value for this draft
- o "YYYY" --> the assigned RFC value for I-D.ietf-netconf-keystore

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- o "2017-06-13" --> the publication date of this draft

The following Appendix section is to be removed prior to publication:

- o Appendix A. Change Log

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 December 15, 2017.

Copyright Notice

Copyright (c) 2017 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	3
1.1. Terminology	3
1.2. Tree Diagrams	4
2. The SSH Client Model	4
2.1. Tree Diagram	4
2.2. Example Usage	5
2.3. YANG Model	6
3. The SSH Server Model	10
3.1. Tree Diagram	10
3.2. Example Usage	11
3.3. YANG Model	11
4. The SSH Common Model	15
4.1. Tree Diagram	15
4.2. Example Usage	15

4.3. YANG Model	16
5. Security Considerations	26
6. IANA Considerations	27
6.1. The IETF XML Registry	27
6.2. The YANG Module Names Registry	28
7. Acknowledgements	28
8. References	28
8.1. Normative References	29
8.2. Informative References	30
Appendix A. Change Log	31
A.1. server-model-09 to 00	31
A.2. 00 to 01	31
A.3. 01 to 02	31
A.4. 02 to 03	31
Authors' Addresses	31

1. Introduction

This document defines three YANG [RFC7950] modules: the first defines a grouping for a generic SSH client, the second defines a grouping for a generic SSH server, and the third defines identities and groupings common to both the client and the server (SSH is defined in [RFC4252], [RFC4253], and [RFC4254]). It is intended that these groupings will be used by applications using the SSH protocol. For instance, these groupings could be used to help define the data model for an OpenSSH [OPENSSSH] server or a NETCONF over SSH [RFC6242] based server.

The client and server YANG modules in this document each define one grouping, which is focused on just SSH-specific configuration, and specifically avoids any transport-level configuration, such as what ports to listen-on or connect-to. This enables applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [RFC8071] could use the grouping for the SSH parts it provides, while adding data nodes for the TCP-level call-home configuration.

1.1. Terminology

The keywords "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].

1.2. Tree Diagrams

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. The SSH Client Model

The SSH client model presented in this section contains one YANG grouping, to just configure the SSH client omitting, for instance, any configuration for which IP address or port the client should connect to.

This grouping references data nodes defined by the keystore model [I-D.ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate which trusted CA certificate a client should use to authenticate X.509v3 certificate based host keys [RFC6187].

2.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-ssh-client module. Please see Section 1.2 for tree diagram notation.

```

module: ietf-ssh-client
groupings:
ssh-client-grouping
  +----- server-auth
  |   +----- trusted-ssh-host-keys?
  |   |       -> /ks:keystore/trusted-host-keys/name
  |   +----- trusted-ca-certs?
  |   |       -> /ks:keystore/trusted-certificates/name
  |   |       {sshcom:ssh-x509-certs}?
  |   +----- trusted-server-certs?
  |   |       -> /ks:keystore/trusted-certificates/name
  |   |       {sshcom:ssh-x509-certs}?
  +----- client-auth
  |   +----- username?          string
  |   +----- (auth-type)?
  |   |   +---:(certificate)
  |   |   |   +----- certificate?  leafref {sshcom:ssh-x509-certs}?
  |   |   +---:(public-key)
  |   |   |   +----- public-key?    -> /ks:keystore/keys/key/name
  |   |   +---:(password)
  |   |   |   +----- password?      string
  +----- transport-params {ssh-client-transport-params-config}?
  |   +----- host-key
  |   |   +----- host-key-alg*    identityref
  |   +----- key-exchange
  |   |   +----- key-exchange-alg* identityref
  |   +----- encryption
  |   |   +----- encryption-alg*  identityref
  |   +----- mac
  |   |   +----- mac-alg*         identityref
  +----- compression
  |   +----- compression-alg*     identityref

```

2.2. Example Usage

This section shows how it would appear if the `ssh-client-grouping` were populated with some data. This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

```
<!-- hypothetical example, as groupings don't have instance data -->
<ssh-client xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-client">

  <!-- which host-keys will this client trust -->
  <server-auth>
    <trusted-ssh-host-keys>explicitly-trusted-ssh-host-keys</trusted-ssh-host-ke
ys>
  </server-auth>

  <!-- how this client will authenticate itself to the server -->
  <client-auth>
    <username>foobar</username>
    <public-key>ex-rsa-key</public-key>
  </client-auth>
</ssh-client>
```

2.3. YANG Model

This YANG module has a normative references to [RFC6991] and [I-D.ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-ssh-client@2017-06-13.yang"

module ietf-ssh-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-client";
  prefix "sshc";

  import ietf-ssh-common {
    prefix sshcom;
    revision-date 2017-06-13; // stable grouping definitions
    reference
      "RFC XXXX: SSH Client and Server Models";
  }

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 6536: Network Configuration Protocol (NETCONF) Access
      Control Model";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC YYYY: Keystore Model";
```



```
}

organization
  "IETF NETCONF (Network Configuration) Working Group";

contact
  "WG Web:    <http://tools.ietf.org/wg/netconf/>
  WG List:    <mailto:netconf@ietf.org>

  Author:     Kent Watsen
               <mailto:kwatsen@juniper.net>

  Author:     Gary Wu
               <mailto:garywu@cisco.com>";

description
  "This module defines a reusable grouping for a SSH client that
  can be used as a basis for specific SSH client instances.

  Copyright (c) 2014 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD
  License set forth in Section 4.c of the IETF Trust's
  Legal Provisions Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";

revision "2017-06-13" {
  description
    "Initial version";
  reference
    "RFC XXXX: SSH Client and Server Models";
}

feature ssh-client-transport-params-config {
  description
    "SSH transport layer parameters are configurable on an SSH
    client.";
}

grouping ssh-client-grouping {
  description
```

```
"A reusable grouping for configuring a SSH client without
any consideration for how an underlying TCP session is
established.";

container server-auth {
  must 'trusted-ssh-host-keys or trusted-ca-certs or trusted-server-certs';
  description
    "Trusted server identities.";
  leaf trusted-ssh-host-keys {
    type leafref {
      path "/ks:keystore/ks:trusted-host-keys/ks:name";
    }
    description
      "A reference to a list of SSH host keys used by the
      SSH client to authenticate SSH server host keys.
      A server host key is authenticate if it is an exact
      match to a configured trusted SSH host key.";
  }

  leaf trusted-ca-certs {
    if-feature sshcom:ssh-x509-certs;
    type leafref {
      path "/ks:keystore/ks:trusted-certificates/ks:name";
    }
    description
      "A reference to a list of certificate authority (CA)
      certificates used by the SSH client to authenticate
      SSH server certificates. A server certificate is
      authenticated if it has a valid chain of trust to
      a configured trusted CA certificate.";
  }

  leaf trusted-server-certs {
    if-feature sshcom:ssh-x509-certs;
    type leafref {
      path "/ks:keystore/ks:trusted-certificates/ks:name";
    }
    description
      "A reference to a list of server certificates used by
      the SSH client to authenticate SSH server certificates.
      A server certificate is authenticated if it is an
      exact match to a configured trusted server certificate.";
  }
}

container client-auth {
  description
    "The credentials used by the client to authenticate to
```

```
        the SSH server.";

    leaf username {
        type string;
        description
            "The username of this user. This will be the username
            used, for instance, to log into an SSH server.";
    }

    choice auth-type {
        description
            "The authentication type.";
        leaf certificate {
            if-feature sshcom:ssh-x509-certs;
            type leafref {
                path "/ks:keystore/ks:keys/ks:key/ks:certificates/"
                    + "ks:certificate/ks:name";
            }
            description
                "A certificates to be used for user authentication.";
        }
        leaf public-key {
            type leafref {
                path "/ks:keystore/ks:keys/ks:key/ks:name";
            }
            description
                "A public keys to be used for user authentication.";
        }
        leaf password {
            nacm:default-deny-all;
            type string;
            description
                "A password to be used for user authentication.";
        }
    }
} // end client-auth

container transport-params {
    if-feature ssh-client-transport-params-config;
    uses sshcom:transport-params-grouping;
    description
        "Configurable parameters for the SSH transport layer.";
}

} // ssh-client-grouping
}
```

<CODE ENDS>

3. The SSH Server Model

The SSH server model presented in this section contains one YANG grouping, for just the SSH-level configuration omitting, for instance, configuration for which ports to open to listen for connections on.

This grouping references data nodes defined by the keystore model [I-D.ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate which host key a server should present.

3.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-ssh-server module. Please see Section 1.2 for tree diagram notation.

```

module: ietf-ssh-server
groupings:
  ssh-server-grouping
    +----- host-keys
    |   +----- host-key* [name]
    |   |   +----- name?          string
    |   |   +----- (host-key-type)
    |   |   |   +---:(public-key)
    |   |   |   |   +----- public-key?    -> /ks:keystore/keys/key/name
    |   |   |   |   +---:(certificate)
    |   |   |   |   +----- certificate?    leafref {sshcom:ssh-x509-certs}?
    |   +----- client-cert-auth {sshcom:ssh-x509-certs}?
    |   |   +----- trusted-ca-certs?
    |   |   |   -> /ks:keystore/trusted-certificates/name
    |   |   +----- trusted-client-certs?
    |   |   |   -> /ks:keystore/trusted-certificates/name
    +----- transport-params {ssh-server-transport-params-config}?
    |   +----- host-key
    |   |   +----- host-key-alg*    identityref
    |   +----- key-exchange
    |   |   +----- key-exchange-alg* identityref
    |   +----- encryption
    |   |   +----- encryption-alg*  identityref
    |   +----- mac
    |   |   +----- mac-alg*         identityref
    |   +----- compression
    |   |   +----- compression-alg*  identityref

```

3.2. Example Usage

This section shows how it would appear if the ssh-server-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

```
<!-- hypothetical example, as groupings don't have instance data -->
<ssh-server xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server">

  <!-- which host-keys will this SSH server present -->
  <host-keys>
    <host-key>
      <name>deployment-specific-certificate</name>
      <certificate>ex-rsa-cert</certificate>
    </host-key>
  </host-keys>

  <!-- NOTE: password/public-key auth is NOT configured here, -->
  <!-- as it is configured in the ietf-system (RFC 7317) -->
  <!-- module instead. -->

  <!-- which client-certs will this SSH server trust -->
  <client-cert-auth>
    <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
    <trusted-client-certs>explicitly-trusted-client-certs</trusted-client-certs>
  </client-cert-auth>

</ssh-server>
```

3.3. YANG Model

This YANG module has a normative references to [RFC4253], [RFC6991], and [I-D.ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-ssh-server@2017-06-13.yang"

module ietf-ssh-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-server";
  prefix "sshs";

  import ietf-ssh-common {
    prefix sshcom;
    revision-date 2017-06-13; // stable grouping definitions
    reference
      "RFC XXXX: SSH Client and Server Models";
  }
}
```

```
}

import ietf-keystore {
  prefix ks;
  reference
    "RFC YYYY: Keystore Model";
}

organization
  "IETF NETCONF (Network Configuration) Working Group";

contact
  "WG Web:    <http://tools.ietf.org/wg/netconf/>
  WG List:    <mailto:netconf@ietf.org>

  Author:     Kent Watsen
               <mailto:kwatsen@juniper.net>";

description
  "This module defines a reusable grouping for a SSH server that
  can be used as a basis for specific SSH server instances.

  Copyright (c) 2014 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD
  License set forth in Section 4.c of the IETF Trust's
  Legal Provisions Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";

revision "2017-06-13" {
  description
    "Initial version";
  reference
    "RFC XXXX: SSH Client and Server Models";
}

// features
feature ssh-server-transport-params-config {
  description
    "SSH transport layer parameters are configurable on an SSH
    server.";
```

```
}

// grouping
grouping ssh-server-grouping {
  description
    "A reusable grouping for configuring a SSH server without
    any consideration for how underlying TCP sessions are
    established.";
  container host-keys {
    description
      "The list of host-keys the SSH server will present when
      establishing a SSH connection.";
    list host-key {
      key name;
      min-elements 1;
      ordered-by user;
      description
        "An ordered list of host keys the SSH server will use to
        construct its ordered list of algorithms, when sending
        its SSH_MSG_KEXINIT message, as defined in Section 7.1
        of RFC 4253.";
      reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
      leaf name {
        type string;
        description
          "An arbitrary name for this host-key";
      }
      choice host-key-type {
        mandatory true;
        description
          "The type of host key being specified";
        leaf public-key {
          type leafref {
            path "/ks:keystore/ks:keys/ks:key/ks:name";
          }
          description
            "The public key is actually identified by the name of
            its cooresponding private-key in the keystore.";
        }
        leaf certificate {
          if-feature sshcom:ssh-x509-certs;
          type leafref {
            path "/ks:keystore/ks:keys/ks:key/ks:certificates/"
              + "ks:certificate/ks:name";
          }
          description
            "The name of a certificate in the keystore.";
        }
      }
    }
  }
}
```

```
    }
  }
}

container client-cert-auth {
  if-feature sshcom:ssh-x509-certs;
  description
    "A reference to a list of trusted certificate authority (CA)
    certificates and a reference to a list of trusted client
    certificates.";
  leaf trusted-ca-certs {
    type leafref {
      path "/ks:keystore/ks:trusted-certificates/ks:name";
    }
    description
      "A reference to a list of certificate authority (CA)
      certificates used by the SSH server to authenticate
      SSH client certificates.";
  }
  leaf trusted-client-certs {
    type leafref {
      path "/ks:keystore/ks:trusted-certificates/ks:name";
    }
    description
      "A reference to a list of client certificates used by
      the SSH server to authenticate SSH client certificates.
      A clients certificate is authenticated if it is an
      exact match to a configured trusted client certificate.";
  }
}

container transport-params {
  if-feature ssh-server-transport-params-config;
  uses sshcom:transport-params-grouping;
  description
    "Configurable parameters for the SSH transport layer.";
}

} // ssh-server-grouping

}
```

<CODE ENDS>

4. The SSH Common Model

The SSH common model presented in this section contains identities and groupings common to both SSH clients and SSH servers. The transport-params-grouping can be used to configure the list of SSH transport algorithms permitted by the SSH client or SSH server. The lists of algorithms are ordered such that, if multiple algorithms are permitted by the client, the algorithm that appears first in its list that is also permitted by the server is used for the SSH transport layer connection. The ability to restrict the the algorithms allowed is provided in this grouping for SSH clients and SSH servers that are capable of doing so and may serve to make SSH clients and SSH servers compliant with security policies.

Features are defined for algorithms that are OPTIONAL or are not widely supported by popular implementations. Note that the list of algorithms is not exhaustive. As well, some algorithms that are REQUIRED by [RFC4253] are missing, notably "ssh-dss" and "diffie-hellman-group1-sha1" due to their weak security and there being alternatives that are widely supported.

4.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-ssh-common module. Please see Section 1.2 for tree diagram notation.

```
module: ietf-ssh-common
  groupings:
    transport-params-grouping
      +----- host-key
      |   +----- host-key-alg*   identityref
      +----- key-exchange
      |   +----- key-exchange-alg*   identityref
      +----- encryption
      |   +----- encryption-alg*   identityref
      +----- mac
      |   +----- mac-alg*   identityref
      +----- compression
      |   +----- compression-alg*   identityref
```

4.2. Example Usage

This section shows how it would appear if the transport-params-grouping were populated with some data.

```
<!-- hypothetical example, as groupings don't have instance data -->
<transport-params xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-common">

  <host-key>
    <host-key-alg>x509v3-rsa2048-sha256</host-key-alg>
    <host-key-alg>ssh-rsa</host-key-alg>
  </host-key>
  <key-exchange>
    <key-exchange-alg>
      diffie-hellman-group-exchange-sha256
    </key-exchange-alg>
  </key-exchange>
  <encryption>
    <encryption-alg>aes256-ctr</encryption-alg>
    <encryption-alg>aes192-ctr</encryption-alg>
    <encryption-alg>aes128-ctr</encryption-alg>
    <encryption-alg>aes256-cbc</encryption-alg>
    <encryption-alg>aes192-cbc</encryption-alg>
    <encryption-alg>aes128-cbc</encryption-alg>
  </encryption>
  <mac>
    <mac-alg>hmac-sha2-256</mac-alg>
    <mac-alg>hmac-sha2-512</mac-alg>
  </mac>
  <compression>
    <compression-alg>none</compression-alg>
  </compression>

</transport-params>
```

4.3. YANG Model

This YANG module has a normative references to [RFC4344], [RFC4419], and [RFC5656].

```
<CODE BEGINS> file "ietf-ssh-common@2017-06-13.yang"

module ietf-ssh-common {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-common";
  prefix "sshcom";

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
```

"WG Web: <<http://tools.ietf.org/wg/netconf/>>
WG List: <<mailto:netconf@ietf.org>>

Author: Kent Watsen
<<mailto:kwatsen@juniper.net>>

Author: Gary Wu
<<mailto:garywu@cisco.com>>" ;

description

"This module defines a common features, identities, and groupings for Secure Shell (SSH).

Copyright (c) 2017 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

```
revision "2017-06-13" {  
  description  
    "Initial version";  
  reference  
    "RFC XXXX: SSH Client and Server Models";  
}  
  
// features  
feature ssh-ecc {  
  description  
    "Elliptic Curve Cryptography is supported for SSH.";  
  reference  
    "RFC 5656: Elliptic Curve Algorithm Integration in the  
      Secure Shell Transport Layer";  
}  
  
feature ssh-x509-certs {  
  description  
    "X.509v3 certificates are supported for SSH as per RFC 6187.";  
  reference  
    "RFC 6187: X.509v3 Certificates for Secure Shell
```

```
        Authentication";
    }

    feature ssh-dh-group-exchange {
        description
            "Diffie-Hellman Group Exchange is supported for SSH.";
        reference
            "RFC 4419: Diffie-Hellman Group Exchange for the
              Secure Shell (SSH) Transport Layer Protocol";
    }

    feature ssh-ctr {
        description
            "SDCTR encryption mode is supported for SSH.";
        reference
            "RFC 4344: The Secure Shell (SSH) Transport Layer
              Encryption Modes";
    }

    feature ssh-sha2 {
        description
            "The SHA2 family of cryptographic hash functions is supported
              for SSH.";
        reference
            "FIPS PUB 180-4: Secure Hash Standard (SHS)";
    }

    feature ssh-zlib {
        description
            "ZLIB (LZ77) compression is supported for SSH.";
        reference
            "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    // identities
    identity public-key-alg-base {
        description
            "Base identity used to identify public key algorithms.";
    }

    identity ssh-dss {
        base public-key-alg-base;
        description
            "Digital Signature Algorithm using SHA-1 as the hashing
              algorithm.";
        reference
            "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }
}
```

```
identity ssh-rsa {
  base public-key-alg-base;
  description
    "RSASSA-PKCS1-v1_5 signature scheme using SHA-1 as the hashing
    algorithm.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity ecdsa-sha2-nistp256 {
  base public-key-alg-base;
  if-feature "ssh-ecc and ssh-sha2";
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
    nistp256 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
    Secure Shell Transport Layer";
}

identity ecdsa-sha2-nistp384 {
  base public-key-alg-base;
  if-feature "ssh-ecc and ssh-sha2";
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
    nistp384 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
    Secure Shell Transport Layer";
}

identity ecdsa-sha2-nistp521 {
  base public-key-alg-base;
  if-feature "ssh-ecc and ssh-sha2";
  description
    "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
    nistp521 curve and the SHA2 family of hashing algorithms.";
  reference
    "RFC 5656: Elliptic Curve Algorithm Integration in the
    Secure Shell Transport Layer";
}

identity x509v3-ssh-rsa {
  base public-key-alg-base;
  if-feature ssh-x509-certs;
  description
    "RSASSA-PKCS1-v1_5 signature scheme using a public key stored in
    an X.509v3 certificate and using SHA-1 as the hashing
```

```
        algorithm.";
    reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
        Authentication";
}

identity x509v3-rsa2048-sha256 {
    base public-key-alg-base;
    if-feature "ssh-x509-certs and ssh-sha2";
    description
        "RSASSA-PKCS1-v1_5 signature scheme using a public key stored in
        an X.509v3 certificate and using SHA-256 as the hashing
        algorithm.  RSA keys conveyed using this format MUST have a
        modulus of at least 2048 bits.";
    reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
        Authentication";
}

identity x509v3-ecdsa-sha2-nistp256 {
    base public-key-alg-base;
    if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
    description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
        nistp256 curve with a public key stored in an X.509v3
        certificate and using the SHA2 family of hashing algorithms.";
    reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
        Authentication";
}

identity x509v3-ecdsa-sha2-nistp384 {
    base public-key-alg-base;
    if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
    description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
        nistp384 curve with a public key stored in an X.509v3
        certificate and using the SHA2 family of hashing algorithms.";
    reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
        Authentication";
}

identity x509v3-ecdsa-sha2-nistp521 {
    base public-key-alg-base;
    if-feature "ssh-ecc and ssh-x509-certs and ssh-sha2";
    description
        "Elliptic Curve Digital Signature Algorithm (ECDSA) using the
```

```
        nistp521 curve with a public key stored in an X.509v3
        certificate and using the SHA2 family of hashing algorithms.";
    reference
        "RFC 6187: X.509v3 Certificates for Secure Shell
        Authentication";
}

identity key-exchange-alg-base {
    description
        "Base identity used to identify key exchange algorithms.";
}

identity diffie-hellman-group14-sha1 {
    base key-exchange-alg-base;
    description
        "Diffie-Hellman key exchange with SHA-1 as HASH and
        Oakley Group 14 (2048-bit MODP Group).";
    reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity diffie-hellman-group-exchange-sha1 {
    base key-exchange-alg-base;
    if-feature ssh-dh-group-exchange;
    description
        "Diffie-Hellman Group and Key Exchange with SHA-1 as HASH.";
    reference
        "RFC 4419: Diffie-Hellman Group Exchange for the
        Secure Shell (SSH) Transport Layer Protocol";
}

identity diffie-hellman-group-exchange-sha256 {
    base key-exchange-alg-base;
    if-feature "ssh-dh-group-exchange and ssh-sha2";
    description
        "Diffie-Hellman Group and Key Exchange with SHA-256 as HASH.";
    reference
        "RFC 4419: Diffie-Hellman Group Exchange for the
        Secure Shell (SSH) Transport Layer Protocol";
}

identity ecdh-sha2-nistp256 {
    base key-exchange-alg-base;
    if-feature "ssh-ecc and ssh-sha2";
    description
        "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
        nistp256 curve and the SHA2 family of hashing algorithms.";
    reference
```

```
        "RFC 5656: Elliptic Curve Algorithm Integration in the
          Secure Shell Transport Layer";
    }

    identity ecdh-sha2-nistp384 {
        base key-exchange-alg-base;
        if-feature "ssh-ecc and ssh-sha2";
        description
            "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
             nistp384 curve and the SHA2 family of hashing algorithms.";
        reference
            "RFC 5656: Elliptic Curve Algorithm Integration in the
             Secure Shell Transport Layer";
    }

    identity ecdh-sha2-nistp521 {
        base key-exchange-alg-base;
        if-feature "ssh-ecc and ssh-sha2";
        description
            "Elliptic Curve Diffie-Hellman (ECDH) key exchange using the
             nistp521 curve and the SHA2 family of hashing algorithms.";
        reference
            "RFC 5656: Elliptic Curve Algorithm Integration in the
             Secure Shell Transport Layer";
    }

    identity encryption-alg-base {
        description
            "Base identity used to identify encryption algorithms.";
    }

    identity triple-des-cbc {
        base encryption-alg-base;
        description
            "Three-key 3DES in CBC mode.";
        reference
            "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity aes128-cbc {
        base encryption-alg-base;
        description
            "AES in CBC mode, with a 128-bit key.";
        reference
            "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity aes192-cbc {
```



```
    base encryption-alg-base;
    description
        "AES in CBC mode, with a 192-bit key.";
    reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity aes256-cbc {
    base encryption-alg-base;
    description
        "AES in CBC mode, with a 256-bit key.";
    reference
        "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

identity aes128-ctr {
    base encryption-alg-base;
    if-feature ssh-ctr;
    description
        "AES in SDCTR mode, with 128-bit key.";
    reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
        Modes";
}

identity aes192-ctr {
    base encryption-alg-base;
    if-feature ssh-ctr;
    description
        "AES in SDCTR mode, with 192-bit key.";
    reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
        Modes";
}

identity aes256-ctr {
    base encryption-alg-base;
    if-feature ssh-ctr;
    description
        "AES in SDCTR mode, with 256-bit key.";
    reference
        "RFC 4344: The Secure Shell (SSH) Transport Layer Encryption
        Modes";
}

identity mac-alg-base {
    description
        "Base identity used to identify message authentication
```

```
        code (MAC) algorithms.";
    }

    identity hmac-sha1 {
        base mac-alg-base;
        description
            "HMAC-SHA1";
        reference
            "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity hmac-sha2-256 {
        base mac-alg-base;
        if-feature "ssh-sha2";
        description
            "HMAC-SHA2-256";
        reference
            "RFC 6668: SHA-2 Data Integrity Verification for the
              Secure Shell (SSH) Transport Layer Protocol";
    }

    identity hmac-sha2-512 {
        base mac-alg-base;
        if-feature "ssh-sha2";
        description
            "HMAC-SHA2-512";
        reference
            "RFC 6668: SHA-2 Data Integrity Verification for the
              Secure Shell (SSH) Transport Layer Protocol";
    }

    identity compression-alg-base {
        description
            "Base identity used to identify compression algorithms.";
    }

    identity none {
        base compression-alg-base;
        description
            "No compression.";
        reference
            "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
    }

    identity zlib {
        base compression-alg-base;
        if-feature ssh-zlib;
        description
```

```
    "ZLIB (LZ77) compression.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
}

// groupings
grouping transport-params-grouping {
  description
    "A reusable grouping for SSH transport parameters.
    For configurable parameters, a zero-element leaf-list of
    algorithms indicates the system default configuration for that
    parameter.";
  reference
    "RFC 4253: The Secure Shell (SSH) Transport Layer Protocol";
  container host-key {
    description
      "Parameters regarding host key.";
    leaf-list host-key-alg {
      type identityref {
        base public-key-alg-base;
      }
      ordered-by user;
      description
        "Host key algorithms in order of descending preference.";
    }
  }
  container key-exchange {
    description
      "Parameters regarding key exchange.";
    leaf-list key-exchange-alg {
      type identityref {
        base key-exchange-alg-base;
      }
      ordered-by user;
      description
        "Key exchange algorithms in order of descending
        preference.";
    }
  }
  container encryption {
    description
      "Parameters regarding encryption.";
    leaf-list encryption-alg {
      type identityref {
        base encryption-alg-base;
      }
      ordered-by user;
      description

```

```

        "Encryption algorithms in order of descending preference.";
    }
}
container mac {
    description
        "Parameters regarding message authentication code (MAC).";
    leaf-list mac-alg {
        type identityref {
            base mac-alg-base;
        }
        ordered-by user;
        description
            "MAC algorithms in order of descending preference.";
    }
}
container compression {
    description
        "Parameters regarding compression.";
    leaf-list compression-alg {
        type identityref {
            base compression-alg-base;
        }
        ordered-by user;
        description
            "Compression algorithms in order of descending preference.";
    }
}
}
}

```

<CODE ENDS>

5. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config)

to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/: The entire data tree defined by this module is sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated 'recovery session'.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

/client-auth/password: This node is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. The best reason for returning this node is to support backup/restore type workflows. This being the case, this node is marked with the NACM value 'default-deny-all'.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

6. IANA Considerations

6.1. The IETF XML Registry

This document registers three URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-ssh-common
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

6.2. The YANG Module Names Registry

This document registers three YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the the following registrations are requested:

name:	ietf-ssh-client
namespace:	urn:ietf:params:xml:ns:yang:ietf-ssh-client
prefix:	sshc
reference:	RFC XXXX
name:	ietf-ssh-server
namespace:	urn:ietf:params:xml:ns:yang:ietf-ssh-server
prefix:	sshs
reference:	RFC XXXX
name:	ietf-ssh-common
namespace:	urn:ietf:params:xml:ns:yang:ietf-ssh-common
prefix:	sshcom
reference:	RFC XXXX

7. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, Balazs Kovacs, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Juergen Schoenwaelder, Phil Shafer, Sean Turner, Michal Vasko, and Bert Wijnen.

8. References

8.1. Normative References

- [I-D.ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-01 (work in progress), March 2017.
- [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>>.
- [RFC4344] Bellare, M., Kohno, T., and C. Namprempre, "The Secure Shell (SSH) Transport Layer Encryption Modes", RFC 4344, DOI 10.17487/RFC4344, January 2006, <<http://www.rfc-editor.org/info/rfc4344>>.
- [RFC4419] Friedl, M., Provos, N., and W. Simpson, "Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4419, DOI 10.17487/RFC4419, March 2006, <<http://www.rfc-editor.org/info/rfc4419>>.
- [RFC5656] Stebila, D. and J. Green, "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer", RFC 5656, DOI 10.17487/RFC5656, December 2009, <<http://www.rfc-editor.org/info/rfc5656>>.
- [RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication", RFC 6187, DOI 10.17487/RFC6187, March 2011, <<http://www.rfc-editor.org/info/rfc6187>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC6668] Bider, D. and M. Baushke, "SHA-2 Data Integrity Verification for the Secure Shell (SSH) Transport Layer Protocol", RFC 6668, DOI 10.17487/RFC6668, July 2012, <<http://www.rfc-editor.org/info/rfc6668>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.

8.2. Informative References

- [OPENSSSH] "OpenSSH", 2016, <<http://www.openssh.com>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <<http://www.rfc-editor.org/info/rfc4252>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<http://www.rfc-editor.org/info/rfc4253>>.
- [RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, DOI 10.17487/RFC4254, January 2006, <<http://www.rfc-editor.org/info/rfc4254>>.
- [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>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.
- [RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home", RFC 8071, DOI 10.17487/RFC8071, February 2017, <<http://www.rfc-editor.org/info/rfc8071>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Added in previously missing ietf-ssh-client module.
- o Noted that '0.0.0.0' and ':::' might have special meanings.

A.2. 00 to 01

- o Renamed "keychain" to "keystore".

A.3. 01 to 02

- o Removed the groupings 'listening-ssh-client-grouping' and 'listening-ssh-server-grouping'. Now modules only contain the transport-independent groupings.
- o Simplified the "client-auth" part in the ietf-ssh-client module. It now inlines what it used to point to keystore for.
- o Added cipher suites for various algorithms into new 'ietf-ssh-common' module.

A.4. 02 to 03

- o Removed 'RESTRICTED' enum from 'password' leaf type.
- o Added a 'must' statement to container 'server-auth' asserting that at least one of the various auth mechanisms must be specified.
- o Fixed description statement for leaf 'trusted-ca-certs'.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Systems

EMail: garywu@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2018

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
VMWare
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
July 2, 2017

Custom Subscription to Event Notifications
draft-ietf-netconf-subscribed-notifications-03

Abstract

This document defines capabilities and operations for the customized establishment of subscriptions upon a publisher's event streams. Also defined are delivery mechanisms for instances of the resulting events. Effectively this allows a subscriber to request and receive a continuous, custom influx of publisher generated information.

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 January 3, 2018.

Copyright Notice

Copyright (c) 2017 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	3
1.1. Motivation	3
1.2. Terminology	4
1.3. Solution Overview	5
2. Solution	6
2.1. Event Streams	6
2.2. Filters	6
2.3. Subscription State Model at the Publisher	7
3. Data Model Trees	8
4. Dynamic Subscriptions	12
4.1. Establishing a Subscription	12
4.2. Modifying a Subscription	13
4.3. Deleting a Subscription	13
4.4. Killing a Subscription	14
5. Configured Subscriptions	14
5.1. Establishing a Configured Subscription	15
5.2. Modifying a Configured Subscription	17
5.3. Deleting a Configured Subscription	17
6. Event (Data Plane) Notifications	18
7. Subscription State Notifications	19
7.1. subscription-started	19
7.2. subscription-modified	20
7.3. subscription-terminated	20
7.4. subscription-suspended	20
7.5. subscription-resumed	20
7.6. notification-complete	20
7.7. replay-complete	21
8. Administrative Functions	21
8.1. Subscription Monitoring	21
8.2. Capability Advertisement	21
8.3. Event Stream Discovery	22
9. Data Model	22
10. Considerations	44
10.1. Implementation Considerations	44
10.2. Security Considerations	44
11. Acknowledgments	45
12. References	45
12.1. Normative References	45
12.2. Informative References	46

Appendix A. Relationships to other drafts	47
A.1. ietf-netconf-netconf-event-notif	47
A.2. ietf-netconf-restconf-notif	47
A.3. ietf-netconf-yang-push	48
A.4. voit-notifications2	48
Appendix B. Issues that are currently being worked and resolved	48
Appendix C. Changes between revisions	48
Authors' Addresses	50

1. Introduction

This document defines capabilities and operations for the customized establishment of subscriptions upon system generated event streams. Also defined are asynchronous delivery mechanisms, where the resulting event instances are placed within notification messages and sent to targeted receivers. Effectively this enables a "Subscribe then Publish" capability where the customized information needs of each target receiver are understood by the publisher before events are marshalled and pushed. The receiver then gets a continuous, custom influx of publisher generated events.

While the functionality defined in this document is transport-agnostic, subscription control plane operations bindings exist for both NETCONF [RFC6241] and RESTCONF [RFC8040]. In addition, bindings for the pushed event instances have been defined for protocols such as NETCONF and HTTP2 [RFC7540]. For specifics on these bindings see [I-D.ietf-netconf-event-notif] and [I-D.ietf-netconf-restconf-notif].

The capabilities and operations defined in this document with implemented in conjunction with [I-D.ietf-netconf-event-notif] are intended to obsolete [RFC5277].

1.1. Motivation

There are various [RFC5277] limitations, many of which have been exposed in [RFC7923] which needed to be solved. Key capabilities supported by this document include:

- o multiple subscriptions on a single transport session
- o support for dynamic and statically configured subscriptions
- o modification of an existing subscription
- o operational counters and instrumentation
- o negotiation of subscription parameters

- o promise theory based interaction model
- o state change notifications (e.g., publisher driven suspension, parameter modification)
- o independence from transport protocol

1.2. 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].

Configured subscription: A subscription installed via a configuration interface which persists across reboots.

Dynamic subscription: A subscription agreed between subscriber and publisher created via RPC subscription state signaling messages.

Event: An occurrence of something that may be of interest. (e.g., a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system.)

Event notification: A set of information intended for a Receiver indicating that one or more Event(s) have occurred. Details of the Event(s) may be included within the Notification.

Filter: Evaluation and/or selection criteria, which may be applied against a targeted set of objects or events in a subscription. Information traverses the filter only if specified filter criteria are met.

NACM: NETCONF Access Control Model.

Publisher: An entity responsible for streaming event notifications per the terms of a Subscription.

Receiver: A target to which a publisher pushes event notifications. For dynamic subscriptions, the receiver and subscriber are the same entity.

Stream (also referred to as "event stream"): A continuous ordered set of events grouped under an explicit criteria.

Subscriber: An entity able to request and negotiate a contract for the generation and push of event notifications from a publisher.

Subscription: A contract with a publisher, stipulating which information one or more receivers wish to have pushed from the publisher without the need for further solicitation.

1.3. Solution Overview

This document describes a transport protocol-agnostic mechanism for subscribing to and receiving event notifications from an event publisher. This mechanism is through the use a subscription.

Two types of subscriptions are supported:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via RPC. If the publisher wants to serve this request, it accepts it, and then starts pushing event notifications. If the publisher does not wish to serve it as requested, then an error response is returned. This response may include hints at subscription parameters which would have been accepted.
2. Configured subscriptions, which allows the management of subscriptions via a configuration interface so that a publisher can send event notifications to configured receiver(s). Support for this capability is optional.

Additional characteristics differentiating configured from dynamic subscriptions include:

- o The lifetime of a dynamic subscription is bounded by transport session used to establish it. For connection-oriented stateful transport like NETCONF, the loss of the transport session will result in the immediate termination of associated dynamic subscriptions. For connectionless or stateless transports like HTTP, it is the lack of receipt acknowledgement of a sequential set of notification messages and/or keep-alives which will terminate dynamic subscriptions. The lifetime of a configured subscription is driven by relevant configuration being present on the running configuration. This implies configured subscriptions persist across reboots, and persist even when transport is unavailable.
- o Configured subscriptions can be modified by any configuration client with write permission on the configuration of the subscription. Dynamic subscriptions can only be modified via an RPC request made upon the original subscribing transport session.

Note that there is no mixing-and-matching of dynamic and configured subscriptions. Specifically, a configured subscription cannot be

modified or deleted using RPC. Similarly, a subscription established via RPC cannot be modified through configuration operations.

The publisher may decide to terminate a dynamic subscription at any time. Similarly, it may decide to temporarily suspend the sending of event notifications for either configured or dynamic subscriptions. Such termination or suspension may be driven by the publisher running out of resources to serve the subscription, or by internal errors on the publisher.

2. Solution

2.1. Event Streams

An event stream is a named entity on a publisher which exposes a continuously updating set of events. Each event stream is available for subscription. It is out of the scope of this document to identify a) how streams are defined, b) how events are defined/generated, and c) how events are assigned to streams.

There are two standardized event streams within this document: NETCONF and SYSLOG. The NETCONF event stream contains all NETCONF XML event information supported by the publisher, except for where it has been explicitly indicated that this the event must be excluded from the NETCONF stream. The SYSLOG event stream mirrors the discrete set entries which are concurrently being placed into a device's local Syslog. Beyond these two, additional streams can be added via model augmentation.

As events are raised by a system, they may be assigned to one or more streams. The event is distributed to receivers where: (1) a subscription includes the identified stream, and (2) subscription filtering allows the event to traverse.

If access control permissions are in use to secure publisher content, then for notifications to be sent to a receiver, that receiver must be allowed access to all the events on the stream. If permissions change during the lifecycle of the of a subscription, then events must be sent or restricted accordingly. This can be be done by re-establishing a subscription with the updated permissions, or by seamlessly updating the permissions of an existing subscription.

2.2. Filters

A publisher implementation MUST support the ability to perform filtering of notification records. Two filtering syntaxes supported are [XPATH] and subtree [RFC6241]. Events which evaluate to "true", or return a non-null selection as a result of the evaluation by the

filter must traverse the filter in their entirety. A subset of information is never stripped from within the event.

2.3. Subscription State Model at the Publisher

Below is the state machine of a subscription for the publisher for a dyanmic subscription. It is important to note that such a subscription doesn't exist at the publisher until it is accepted and made active. The mere request by a subscriber to establish a subscription is insufficient for that asserted subscription to be externally visible via this state machine.

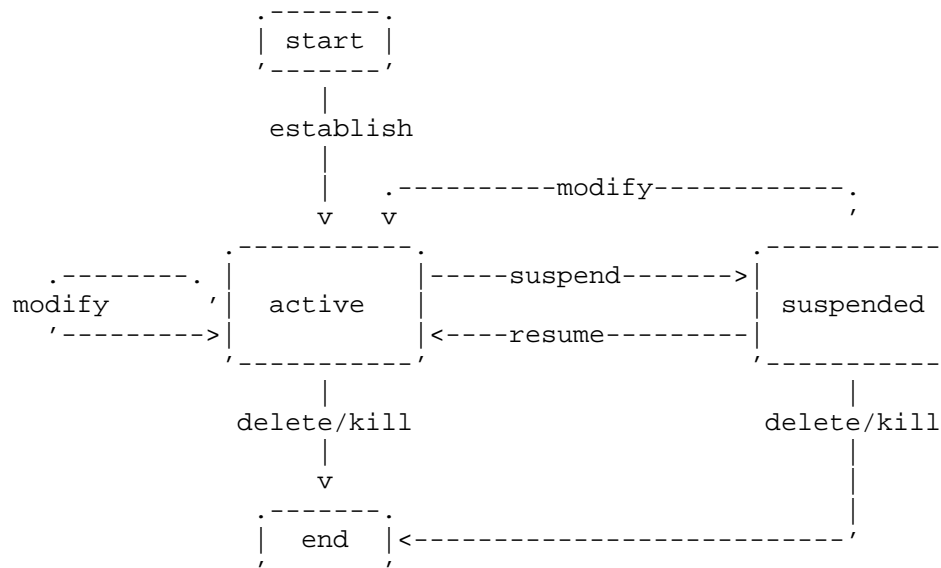


Figure 1: Subscription states at publisher

Of interest in this state machine are the following:

- o Successful establish or modify RPCs put the subscription into an active state.
- o Failed modify RPCs will leave the subscription in its previous state, with no visible change to any streaming updates.
- o A delete or kill RPC will end the subscription.
- o Suspend and resume state changes are driven by internal process and prioritization. There are no external controls over suspend and resume.

An equivalent state machine exists for configured subscriptions. However the transition between states is via configuration operations rather than via RPC.

3. Data Model Trees

```

module: ietf-subscribed-notifications
  +--ro streams
  |   +--ro stream*   stream
  +--rw filters
  |   +--rw filter* [identifier]
  |   |   +--rw identifier          filter-id
  |   |   +--rw (filter-type)?
  |   |   |   +--:(event-filter)
  |   |   |   |   +--rw event-filter-type    event-filter-type
  |   |   |   |   +--rw event-filter
  |   +--rw subscription-config {configured-subscriptions}?
  |   |   +--rw subscription* [identifier]
  |   |   |   +--rw identifier          subscription-id
  |   |   |   +--rw encoding?          encoding
  |   |   |   +--rw (target)
  |   |   |   |   +--:(event-stream)
  |   |   |   |   |   +--rw stream          stream
  |   |   |   +--rw (applied-filter)
  |   |   |   |   +--:(by-reference)
  |   |   |   |   |   +--rw filter-ref          filter-ref
  |   |   |   +--:(within-subscription)
  |   |   |   |   +--rw (filter-type)?
  |   |   |   |   |   +--:(event-filter)
  |   |   |   |   |   |   +--rw event-filter-type    event-filter-type
  |   |   |   |   |   |   +--rw event-filter
  |   |   |   +--rw stop-time?          yang:date-and-time
  |   |   +--rw receivers
  |   |   |   +--rw receiver* [address port]
  |   |   |   |   +--rw address          inet:host
  |   |   |   |   +--rw port            inet:port-number
  |   |   |   |   +--rw protocol?      transport-protocol
  |   |   |   +--rw (notification-origin)?
  |   |   |   |   +--:(interface-originated)
  |   |   |   |   |   +--rw source-interface?    if:interface-ref
  |   |   |   |   +--:(address-originated)
  |   |   |   |   |   +--rw source-vrf?          string
  |   |   |   |   |   +--rw source-address      inet:ip-address-no-zone
  |   +--ro subscriptions
  |   |   +--ro subscription* [identifier]
  |   |   |   +--ro identifier          subscription-id
  |   |   |   +--ro configured-subscription?
  |   |   |   |   empty {configured-subscriptions}?

```

```

+--ro encoding?                encoding
+--ro (target)
|   +--:(event-stream)
|       +--ro stream                stream
|       +--ro replay-start-time?    yang:date-and-time {replay}?
+--ro (applied-filter)
|   +--:(by-reference)
|       |   +--ro filter-ref        filter-ref
|   +--:(within-subscription)
|       +--ro (filter-type)?
|           +--:(event-filter)
|               +--ro event-filter-type    event-filter-type
|               +--ro event-filter
+--ro stop-time?                yang:date-and-time
+--ro (notification-origin)?
|   +--:(interface-originated)
|       |   +--ro source-interface?    if:interface-ref
|   +--:(address-originated)
|       +--ro source-vrf?              string
|       +--ro source-address            inet:ip-address-no-zone
+--ro receivers
    +--ro receiver* [address port]
        +--ro address                inet:host
        +--ro port                    inet:port-number
        +--ro protocol?               transport-protocol
        +--ro pushed-notifications?   yang:counter64
        +--ro excluded-notifications? yang:counter64
        +--ro status                   subscription-status

rpcs:
+---x establish-subscription
|   +---w input
|       +---w encoding?                encoding
|       +---w (target)
|           +--:(event-stream)
|               +---w stream                stream
|               +---w replay-start-time?    yang:date-and-time {replay}?
|       +---w (applied-filter)
|           +--:(by-reference)
|               |   +---w filter-ref        filter-ref
|           +--:(within-subscription)
|               +---w (filter-type)?
|                   +--:(event-filter)
|                       +---w event-filter-type    event-filter-type
|                       +---w event-filter
|       +---w stop-time?                yang:date-and-time
+---ro output
    +--ro subscription-result          subscription-result

```

```

|      +--ro (result)?
|      |      +---:(no-success)
|      |      |      +--ro filter-failure?          string
|      |      |      +--ro replay-start-time-hint?   yang:date-and-time
|      |      +---:(success)
|      |      +--ro identifier                        subscription-id
+---x modify-subscription
|   +---w input
|   |   +---w identifier?                            subscription-id
|   |   +---w (applied-filter)
|   |   |   +---:(by-reference)
|   |   |   |   +---w filter-ref                    filter-ref
|   |   |   +---:(within-subscription)
|   |   |   +---w (filter-type)?
|   |   |   |   +---:(event-filter)
|   |   |   |   +---w event-filter-type            event-filter-type
|   |   |   |   +---w event-filter
|   |   +---w stop-time?                            yang:date-and-time
+---ro output
|   +--ro subscription-result                        subscription-result
|   +--ro (result)?
|   |   +---:(no-success)
|   |   +--ro filter-failure?                        string
+---x delete-subscription
|   +---w input
|   |   +---w identifier                            subscription-id
+---ro output
|   +--ro subscription-result                        subscription-result
+---x kill-subscription
|   +---w input
|   |   +---w identifier                            subscription-id
+---ro output
|   +--ro subscription-result                        subscription-result

notifications:
+---n replay-complete
|   +--ro identifier                                subscription-id
+---n notification-complete
|   +--ro identifier                                subscription-id
+---n subscription-started
|   +--ro identifier                                subscription-id
|   +--ro encoding?                                encoding
|   +--ro (target)
|   |   +---:(event-stream)
|   |   |   +--ro stream                            stream
|   |   |   +--ro replay-start-time?                yang:date-and-time {replay}?
+---ro (applied-filter)
|   +---:(by-reference)

```

```

| | | +--ro filter-ref          filter-ref
| | | +---:(within-subscription)
| | | | +--ro (filter-type)?
| | | | | +---:(event-filter)
| | | | | | +--ro event-filter-type    event-filter-type
| | | | | | +--ro event-filter
| | | +--ro stop-time?          yang:date-and-time
+---n subscription-resumed
| | +--ro identifier            subscription-id
+---n subscription-modified
| | +--ro identifier            subscription-id
| | +--ro encoding?             encoding
| | +--ro (target)
| | | +---:(event-stream)
| | | | +--ro stream            stream
| | | | +--ro replay-start-time? yang:date-and-time {replay}?
+--ro (applied-filter)
| | +---:(by-reference)
| | | +--ro filter-ref          filter-ref
| | | +---:(within-subscription)
| | | | +--ro (filter-type)?
| | | | | +---:(event-filter)
| | | | | | +--ro event-filter-type    event-filter-type
| | | | | | +--ro event-filter
| | +--ro stop-time?          yang:date-and-time
+---n subscription-terminated
| | +--ro identifier            subscription-id
| | +--ro error-id              subscription-errors
| | +--ro filter-failure?       string
+---n subscription-suspended
| | +--ro identifier            subscription-id
| | +--ro error-id              subscription-errors
| | +--ro filter-failure?       string

```

The top-level decompositions of data model are as follows:

- o "Streams" contains a list of event streams that are supported by the publisher and against which subscription is allowed.
- o "Filters" contains a configurable list of filters that can be applied to a subscription. This allows users to reference an existing filter definition as an alternative to defining a filter inline for each subscription.
- o "Subscription-config" contains the configuration of configured subscriptions. The parameters of each configured subscription are a superset of the parameters of a dynamic subscription and use the same groupings. In addition, the configured subscriptions must

also specify intended receivers and may specify the push source from which to send the stream of notification messages.

- o "Subscriptions" contains a list of all subscriptions on a publisher, both configured and dynamic. It can be used to retrieve information about the subscriptions which a publisher is serving.

The data model also contains a number of notifications that allow a publisher to signal information about a subscription. Finally, the data model contains a number of RPC definitions that are used to manage dynamic subscriptions.

4. Dynamic Subscriptions

Dynamic subscriptions are managed via RPC.

4.1. Establishing a Subscription

The <establish-subscription> operation allows a subscriber to request the creation of a subscription via RPC.

The input parameters of the operation are:

- o A stream which identifies the domain of events against which the subscription is applied.
- o A filter which may reduce the set of events pushed.
- o The desired encoding for the returned events. By default, updates are encoded using XML. Other encodings may be supported, such as JSON.
- o An optional stop time for the subscription.
- o An optional start time which indicates that this subscription is requesting a replay push of events previously generated.

If the publisher cannot satisfy the <establish-subscription> request, it sends a negative <subscription-result> element. If the subscriber has no authorization to establish the subscription, the <subscription-result> indicates an authorization error. Optionally, the <subscription-result> may include one or more hints on alternative input parameters and value which would have resulted in an accepted subscription.

Subscription requests must fail if a filter with invalid syntax is provided or if the name of a non-existent stream is provided.

4.1.1. Replay Subscription

Only viable for dynamic subscriptions made on event streams, if the replay feature is supported, a subscription may request that previously generated events be sent. These would then be followed by events generated after the subscription is established.

The presence of a start time is the indicator that there is requested replay for this subscription. The start time must be earlier than the current time. If the start time points earlier than the maintained history of Publisher's event buffer, then the subscription must be rejected. In this case the error response to the <establish-subscription> request should include a start time supportable by the Publisher.

4.2. Modifying a Subscription

The <modify-subscription> operation permits changing the terms of an existing dynamic subscription previously established on that transport session. Subscriptions created by configuration operations cannot be modified via this RPC. Dynamic subscriptions can be modified one or multiple times. If the publisher accepts the requested modifications, it immediately starts sending events based on the new terms, completely ignoring the previous ones. If the publisher rejects the request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. The contents of a such a rejected modification may include one or more hints on alternative input parameters and value which would have resulted in a successfully modified subscription.

Dynamic subscriptions established via RPC can only be modified via RPC using the same transport session used to establish that subscription.

4.3. Deleting a Subscription

The <delete-subscription> operation permits canceling an existing subscription previously established on that transport session. If the publisher accepts the request, it immediately stops sending events for the subscription. If the publisher rejects the request, all subscriptions remain as prior to the request. That is, the request has no impact whatsoever.

Subscriptions established via RPC can only be deleted via RPC using the same transport session used for subscription establishment. Configured subscriptions cannot be deleted using RPCs. Instead, configured subscriptions are deleted as part of regular configuration

operations. Publishers MUST reject any RPC attempt to delete configured subscriptions.

4.4. Killing a Subscription

The <kill-subscription> operation permits an operator to end any dynamic subscription. The publisher must accept the request for any dynamic subscription, and immediately stop sending events.

Configured subscriptions cannot be kill using this RPC. Instead, configured subscriptions are deleted as part of regular configuration operations. Publishers MUST reject any RPC attempt to kill a configured subscription.

5. Configured Subscriptions

A configured subscription is a subscription installed via a configuration interface.

Configured subscriptions persist across reboots, and persist even when transport is unavailable.

Configured subscriptions can be modified by any configuration client with write permissions for the configuration of the subscription. Subscriptions can be modified or terminated via the configuration interface at any point of their lifetime.

Supporting configured subscriptions is optional and advertised using the "configured-subscriptions" feature.

In addition to subscription parameters that apply to dynamic subscriptions, the following additional parameters apply to configured subscriptions:

- o One or more receiver IP addresses (and corresponding ports) intended as the destination for push updates for each subscription. In addition, the transport protocol for each destination may be defined.
- o Optional parameters to identify an egress interface or IP address / VRF where a subscription updates should be pushed from the publisher. Where these are not explicitly included, push updates should egress the publisher's default interface having reachability to a receiver.

5.1. Establishing a Configured Subscription

Configured subscriptions are established using configuration operations against the top-level subtree subscription-config. There are two key differences between RPC and <edit-config> RPC operations for subscription establishment. Firstly, <edit-config> operations install a subscription without question, while RPCs may support negotiation and rejection of requests. Secondly, while RPCs mandate that the subscriber establishing the subscription is the only receiver of the notifications, <edit-config> operations permit specifying receivers independent of any tracked subscriber. Because there is no explicit association with an existing transport session, <edit-config> operations require additional parameters beyond those of dynamic subscriptions to indicate the receivers of the notifications and possibly the source of the notifications such as a specific egress interface.

Immediately after a subscription is successfully established, the publisher sends to identified receivers a control-plane notification stating the subscription has been established (subscription-started).

It is quite possible that upon configuration, reboot, or even steady-state operations, a transport session may not be currently available to the receiver. In this case, when there is something to transport for an active subscription, transport protocol specific call-home operations will be used to establish the connection.

As an example at subscription establishment using <edit-config> over NETCONF, a client may send:


```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <subscription-config
      xmlns="urn:ietf:params:xml:ns:netconf:notification:2.0">
      <subscription>
        <subscription-id>
          1922
        </subscription-id>
        <stream>
          foo
        </stream>
        <receiver>
          <address>
            1.2.3.4
          </address>
          <port>
            1234
          </port>
        </receiver>
      </subscription>
    </subscription-config>
  </edit-config>
</rpc>
```

Figure 2: Configured subscription creation via NETCONF

if the request is accepted, the publisher would reply:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

Figure 3: Successful NETCONF configured subscription response

if the request is not accepted because the publisher cannot serve it, the publisher may reply:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>resource-denied</error-tag>
    <error-severity>error</error-severity>
    <error-message xml:lang="en">
      Temporarily the publisher cannot serve this
      subscription due to the current workload.
    </error-message>
  </rpc-error>
</rpc-reply>
```

Figure 4: A NETCONF response for a failed configured subscription creation

5.2. Modifying a Configured Subscription

Configured subscriptions can be modified using configuration operations against the top-level subtree subscription-config.

Immediately after a subscription is successfully modified, the publisher sends to the existing receivers a control-plane notification stating the subscription has been modified (i.e., subscription-modified).

If the modification involved adding and/or removing receivers, those modified receivers are sent control-plane notifications, indicating they have been added (i.e., subscription-started to a specific receiver) or removed (i.e., subscription-terminated to a specific receiver.)

5.3. Deleting a Configured Subscription

Subscriptions can be deleted using configuration operations against the top-level subtree subscription-config. For example, in RESTCONF:

```
DELETE /subscription-config/subscription=1922 HTTP/1.1
Host: example.com

HTTP/1.1 204 No Content
Date: Sun, 24 Jul 2016 11:23:40 GMT
Server: example-server
```

Figure 5: Deleting a configured subscription

Immediately after a subscription is successfully deleted, the publisher sends to all receivers of that subscription a control-plane

notification stating the subscription has been terminated (subscription-terminated).

6. Event (Data Plane) Notifications

Once a subscription has been set up, the publisher streams (asynchronously) notifications per the terms of the subscription. We refer to these as event notifications. For dynamic subscriptions set up via RPC operations, event notifications are sent over the session used to establish the subscription. For configured subscriptions, event notifications are sent over the specified connections.

An event notification is sent to a receiver when something of interest occurs which is able to traverse all specified filtering and access control criteria. At a minimum this event notification must include:

- o a subscription-id element of type uint32 which corresponds to the responsible subscription in the Publisher.
- o a timestamp indicating when event was identified and recorded by the event source. This timestamp must support the indication of time zone. The default timestamp is the eventTime element of type dateTime and compliant to [RFC3339]. Additional timestamp elements and formats are outside the scope of this document.
- o the event notification content tagged and provided by a source in the publisher.

Additional header and event bundling capabilities not defined in this document may transparently be included within the event notification.

The following is an example of a compliant event notification. This example extending the example within [RFC7950] section 7.16.3 to include the mandatory information described above:

```
<notification
  xmlns=" urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-id>500</subscription-id>
  <link-failure xmlns="http://acme.example.com/system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>
```

Figure 6: Data plane notification

While this extended [RFC7950] section 7.16 notification provides a valid method of encapsulating subscribed notifications, other transport encapsulation methods are also viable. Improvements may be achieved in some implementations in the following ways:

- o transport efficiency may be gained by allowing the encapsulation and bundled push of multiple events within the same event notification.
- o identifiers to designate the current and previous event notification can be used to discover duplicated and dropped notifications
- o additional header types can be used to pass relevant metadata.
- o a signature or hash can be included to verify the efficacy of the Publisher

This is being explored in NETMOD Notifications 2.0 [I-D.void-notifications2].

7. Subscription State Notifications

In addition to data plane notifications, a publisher may send subscription state notifications to indicate to receivers that an event related to the subscription management has occurred.

Subscription state notifications are unlike other notifications in that they are not general-purpose notifications. They cannot be filtered out, and they are delivered only to directly impacted receiver(s) of a subscription. The definition of subscription state notifications is distinct from other notifications by making use of a YANG extension tagging them as subscription state notification.

Subscription state notifications include indications that a replay of events has been completed, that a subscription is done because an end time has been reached, and that a subscription has started, been modified, been terminated, or been suspended. They are described in the following subsections.

7.1. subscription-started

This notification indicates that a configured subscription has started and data updates are beginning to be sent. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

7.2. subscription-modified

This notification indicates that a configured subscription has been modified successfully. This notification includes the parameters of the subscription, except for the receiver(s) addressing information and push-source information. Note that for RPC-based subscriptions, no such notifications are sent.

7.3. subscription-terminated

This notification indicates that a subscription has been terminated by the publisher. The notification includes the reason for the termination. The publisher may decide to terminate a subscription when it is running out of resources for serving it, an internal error occurs, etc. Publisher-driven terminations are notified to all receivers. The management plane can also terminate configured subscriptions using configuration operations.

Subscribers can terminate via RPC subscriptions established via a delete-subscription RPC. In such cases, no subscription-terminated notifications are sent. However if a kill-subscription RPC is sent, or some other event results in the end of a subscription, then there must be a notification that the subscription has been ended.

7.4. subscription-suspended

This notification indicates that a publisher has suspended a subscription. The notification includes the reason for the suspension. A possible reason is the lack of resources to serve it. No further data plane notifications will be sent until the subscription resumes. Suspensions are notified to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

7.5. subscription-resumed

This notification indicates that a previously suspended subscription has been resumed. Data plane notifications generated in the future will be sent after the subscription terms. These notifications go to the subscriber (in the case of dynamic subscriptions) and all receivers (in the case of configured subscriptions).

7.6. notification-complete

This notification is sent to indicate that a subscription, which includes a stop time, has finished passing events.

7.7. replay-complete

This notification indicates that all of the events prior to the current time have been sent. This includes new events generated since the start of the subscription. This notification must not be sent for any other reason.

If subscription contains no stop time, or has a stop time which has not been reached, then after the replay-complete notification has been sent events will be sent in sequence as they arise naturally within the system.

8. Administrative Functions

8.1. Subscription Monitoring

Container "subscriptions" in the YANG module below contains the state of all known subscriptions. This includes subscriptions that were established (and have not yet been deleted) using RPCs, as well as subscriptions that have been configured as part of configuration. Using the <get> operation with NETCONF, or subscribing to this information via [I-D.ietf-netconf-yang-push] allows the status of subscriptions to be monitored.

Each subscription is represented as a list element. The associated information includes an identifier for the subscription, receiver counter information, the status of the subscription, as well as the various subscription parameters that are in effect. The subscription status indicates the subscription's state with each receiver (e.g., is currently active or suspended). Leaf "configured-subscription" indicates whether the subscription came into being via configuration or via RPC.

Subscriptions that were established by RPC are removed from the list once they expire (reaching stop-time) or when they are terminated. Subscriptions that were established by configuration need to be deleted from the configuration by a configuration editing operation even if the stop time has been passed.

8.2. Capability Advertisement

Capabilities are advertised in messages sent by each peer during session establishment [RFC6241]. Publishers supporting the RPCs and Notifications in this document must advertise the capability "urn:ietf:params:netconf:capability:notification:2.0".

If a subscriber only supports [RFC5277] and not this specification, then they will recognize the capability

"urn:ietf:params:netconf:capability:notification:1.0" and ignore the capability defined in this document.

8.3. Event Stream Discovery

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. A client can retrieve this list like any other YANG-defined data, for example using the <get> operation when using NETCONF.

9. Data Model

```
<CODE BEGINS> file "ietf-subscribed-notifications.yang"
module ietf-subscribed-notifications {
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications";

  prefix sn;

  import ietf-yang-types {
    prefix yang;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import ietf-interfaces {
    prefix if;
  }

  organization "IETF";
  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>

    WG Chair: Mahesh Jethanandani
               <mailto:mjethanandani@gmail.com>

    WG Chair: Mehmet Ersue
               <mailto:mehmet.ersue@nokia.com>

    Editor:    Alexander Clemm
               <mailto:ludwig@clemm.org>

    Editor:    Eric Voit
               <mailto:evoit@cisco.com>

    Editor:    Alberto Gonzalez Prieto
```

<mailto:agonzalezpri@vmware.com>

Editor: Einar Nilsen-Nygaard
<mailto:einarnn@cisco.com>

Editor: Ambika Prasad Tripathy
<mailto:ambtripa@cisco.com>;

description

"This module contains conceptual YANG specification for
subscribing to events an receiving event notifications.";

revision 2017-05-31 {

description

"Filtering and stream structures updated, replay a feature.";

reference

"draft-ietf-netconf-subscribed-notifications-03";

}

/*

* FEATURES

*/

feature json {

description

"This feature indicates that JSON encoding of notifications
is supported.";

}

feature configured-subscriptions {

description

"This feature indicates that management plane configuration
of subscription is supported.";

}

feature replay {

description

"This feature indicates that historical event replay is
supported. With replay, it is possible for past events to be
will be streamed in chronological order.";

}

/*

* EXTENSIONS

*/

extension subscription-state-notif {


```
description
  "This statement applies only to notifications. It indicates that
  the notification is a subscription state notification (aka OAM
  notification). Therefore it does not participate in a regular
  event stream and does not need to be specifically subscribed
  in order to receive notifications."
}

/*
 * IDENTITIES
 */

/* Identities for streams */
identity stream {
  description
    "Base identity to represent a generic stream of event
    notifications exposed for subscription by a system."
}
identity NETCONF {
  base stream;
  description
    "Default NETCONF event stream, containing events based on
    notifications defined as YANG modules that are supported by the
    system. As a historical reference, this contains the same set
    of events in a default RFC-5277 NETCONF stream."
}
identity SYSLOG {
  base stream;
  description
    "A stream of events mirroring the discrete set entries
    concurrently being placed into a device's local Syslog."
}
identity custom-stream {
  base stream;
  description
    "A supported stream not defined via an identity in this model"
}

/* Identities for event filters */

identity event-filter {
  description
    "Evaluation criteria used as a pass/fail test against events.
    If a filter element is specified to look for data of a particular
    value, and the data item is not present within a particular event
    for its value to be checked against, the event will be filtered
    out. For example, if one were to check for 'severity=critical' in
    an event where this object does not exist, then the event would
```

```
    not traverse.";
}
identity subtree-event-filter {
  base event-filter;
  description
    "An RFC-6241 based filter which attempts to select nodes within an
    event. After evaluation, the return of a non-empty node set means
    that the filter is successfully traversed.";
  reference "RFC-6241, #5.1";
}
identity xpath-event-filter {
  base event-filter;
  description
    "A filter applied to an event which follows the syntax specified
    in yang:xpath1.0. Success is indicated by either a positive
    boolean result, or a non-null node selection.";
  reference "XPath: http://www.w3.org/TR/1999/REC-xpath-19991116";
}

/* Identities for subscription results */
identity subscription-result {
  description
    "Base identity for RPC responses and State Change Notifications
    providing information on the creation, modification, deletion of
    subscriptions.";
}
identity ok {
  base subscription-result;
  description
    "OK - RPC was successful and was performed as requested.";
}
identity error {
  base subscription-result;
  description
    "Problem with subscription. Base identity for error return
    codes for RPCs and State Change Notifications.";
}

/* Identities for subscription stream status */
identity subscription-status {
  description
    "Base identity for the status of subscriptions and datastreams.";
}
identity active {
  base subscription-status;
  description
    "Status is active and healthy.";
}
```

```
identity inactive {
  base subscription-status;
  description
    "Status is inactive, for example after the stop time, but not
    yet deleted from the configuration.";
}
identity suspended {
  base subscription-status;
  description
    "The status is suspended, meaning that the publisher is currently
    unable to provide the negotiated updates for the subscription.";
}
identity in-error {
  base subscription-status;
  description
    "The status is in error or degraded, meaning that stream and/or
    subscription is currently unable to provide the negotiated
    notifications.";
}

/* Identities for subscription errors */

identity internal-error {
  base error;
  description
    "Error within publisher prohibits operation.";
}
identity suspension-timeout {
  base error;
  description
    "Termination of previously suspended subscription. The publisher
    has eliminated the subscription as it exceeded a time limit for
    suspension.";
}
identity stream-unavailable {
  base error;
  description
    "Stream does not exist or is not available to the receiver.";
}
identity encoding-unavailable {
  base error;
  description
    "Encoding not supported";
}
identity replay-unsupported {
  base error;
  description
    "Replay cannot be performed for this subscription. The publisher
```

```
    does not provide the requested historic information via replay.";
}
identity history-unavailable {
    base error;
    description
        "Replay request too far into the past. The publisher does store
        historic information for all parts of requested subscription, but
        not back to the requested timestamp.";
}
identity filter-unavailable {
    base error;
    description
        "Referenced filter does not exist";
}
identity filter-type-unsupported {
    base error;
    description
        "Cannot parse syntax within the filter.";
}
identity filter-unsupported {
    base error;
    description
        "Failure can be from a syntax error, or a syntax too complex to be
        processed by the platform. The supplemental info should include
        the invalid part of the filter.";
}
identity namespace-unavailable {
    base error;
    description
        "Referenced namespace doesn't exist or is unavailable
        to the receiver.";
}
identity no-such-subscription {
    base error;
    description
        "Referenced subscription doesn't exist. This may be as a result of
        a non-existent subscription ID, an ID which belongs to another
        subscriber, or an ID for acceptable subscription which has been
        statically configured.";
}
identity error-insufficient-resources {
    base error;
    description
        "The server has insufficient resources to support the
        subscription as requested by an RPC.";
}
identity unsupportable-volume {
    base error;
```

```
    description
        "The publisher cannot support the volume of information intended
        to be sent for an existing subscription.";
}
identity error-no-such-option {
    base error;
    description
        "A requested parameter setting is not supported.";
}

/* Identities for encodings */
identity encodings {
    description
        "Base identity to represent data encodings";
}
identity encode-xml {
    base encodings;
    description
        "Encode data using XML";
}
identity encode-json {
    base encodings;
    description
        "Encode data using JSON";
}

/* Identities for transports */
identity transport {
    description
        "An identity that represents a transport protocol for event
        notifications";
}
identity netconf {
    base transport;
    description
        "Netconf notifications as a transport.";
}
identity http2 {
    base transport;
    description
        "HTTP2 notifications as a transport";
}

/*
 * TYPEDEFS
 */

typedef subscription-id {
```

```
    type uint32;
    description
        "A type for subscription identifiers.";
}

typedef filter-id {
    type uint32;
    description
        "A type to identify filters which can be associated with a
        subscription.";
}

typedef subscription-result {
    type identityref {
        base subscription-result;
    }
    description
        "The result of a subscription operation";
}

typedef subscription-errors {
    type identityref {
        base error;
    }
    description
        "The reason for the failure of an RPC request or the sending
        of a subscription suspension or termination notification";
}

typedef encoding {
    type identityref {
        base encodings;
    }
    description
        "Specifies a data encoding, e.g. for a data subscription.";
}

typedef event-filter-type {
    type identityref {
        base event-filter;
    }
    description
        "Specifies a known type of event filter.";
}

typedef subscription-status {
    type identityref {
        base subscription-status;
    }
}
```

```
    }
    description
      "Specifies the status of a subscription.";
  }

  typedef transport-protocol {
    type identityref {
      base transport;
    }
    description
      "Specifies transport protocol used to send notifications to a
      receiver.";
  }

  typedef notification-origin {
    type enumeration {
      enum "interface-originated" {
        description
          "Notifications will be sent from a specific interface on a
          publisher";
      }
      enum "address-originated" {
        description
          "Notifications will be sent from a specific address on a
          publisher";
      }
    }
    description
      "Specifies from where notifications will be sourced when
      being sent by the publisher.";
  }

  typedef stream {
    type identityref {
      base stream;
    }
    description
      "Specifies a system-provided datastream.";
  }

  typedef filter-ref {
    type leafref {
      path "/sn:filters/sn:filter/sn:identifier";
    }
    description
      "This type is used to reference a filter.";
  }
```

```
/*
 * GROUPINGS
 */

grouping base-filter {
  description
    "This grouping defines the base for filters for notification
    events.";
  choice filter-type {
    description
      "A filter needs to be a single filter of a given type. Mixing
      and matching of multiple filters does not occur at the level of
      this grouping.";
    case event-filter {
      leaf event-filter-type {
        type event-filter-type;
        mandatory true;
        description
          "A filter needs to be a known and understood syntax if it is
          to be interpretable by a device.";
      }
      anyxml event-filter {
        mandatory true;
        description
          "Event stream evaluation criteria encoded in a syntax of a
          supported type of filter. If the filter is applied
          against an event stream and there is a non-empty or
          positive result, the event is passed along.";
      }
    }
  }
}

grouping subscription-policy-non-modifiable {
  description
    "This grouping describes the information in a subscription which
    should not change during the life of the subscription.";
  leaf encoding {
    type encoding;
    default "encode-xml";
    description
      "The type of encoding for the subscribed data. Default is XML";
  }
  choice target {
    mandatory true;
    description
      "A filter must be applied against some source of information.
      This identifies the target for the filter.";
  }
}
```



```
    case event-stream {
      leaf stream {
        type stream;
        mandatory true;
        description
          "Indicates a stream of events against which to apply
           a filter.";
      }
    }
  }
}

grouping subscription-policy-modifiable {
  description
    "This grouping describes all objects which may be changed
     in a subscription via an RPC.";
  choice applied-filter {
    mandatory true;
    description
      "A filter must be applied to a subscription.  And that filter
       will come either referenced from a global list, or be provided
       within the subscription itself.";
    case by-reference {
      description
        "Incorporate a filter that has been configured separately.";
      leaf filter-ref {
        type filter-ref;
        mandatory true;
        description
          "References an existing filter which is to be applied to
           the subscription.";
      }
    }
    case within-subscription {
      uses base-filter;
      description
        "Local definition allows a filter to have the same lifecycle
         as the subscription.";
    }
  }
}
leaf stop-time {
  type yang:date-and-time;
  description
    "Identifies a time after which notification events should not
     be sent.  If stop-time is not present, the notifications will
     continue until the subscription is terminated.  If
     replay-start-time exists, stop-time must for a subsequent time.
     If replay-start-time doesn't exist, stop-time must for a future
```

```
        time.";
    }
}

grouping subscription-policy {
    description
        "This grouping describes information concerning a subscription.";
    uses subscription-policy-non-modifiable {
        augment target/event-stream {
            description
                "Adds additional objects which must be set just by RPC.";
            leaf replay-start-time {
                if-feature "replay";
                type yang:date-and-time;
                description
                    "Used to trigger the replay feature and indicate that the
                     replay should start at the time specified.  If
                     replay-start-time is not present, this is not a replay
                     subscription and event pushes should start immediately.  It
                     is never valid to specify start times that are later than
                     or equal to the current time.";
            }
        }
    }
}
uses subscription-policy-modifiable;
}

grouping notification-origin-info {
    description
        "Defines the sender source from which notifications for a
         configured subscription are sent.";
    choice notification-origin {
        description
            "Identifies the egress interface on the Publisher from which
             notifications will or are being sent.";
        case interface-originated {
            description
                "When the push source is out of an interface on the
                 Publisher established via static configuration.";
            leaf source-interface {
                type if:interface-ref;
                description
                    "References the interface for notifications.";
            }
        }
        case address-originated {
            description
                "When the push source is out of an IP address on the
```

```
        Publisher established via static configuration.";
    leaf source-vrf {
        type string;
        description
            "Network instance name for the VRF. This could also have
            been a leafref to draft-ietf-rtgwg-ni-model, but that model
            is not complete, and might not be implemented on a box.";
    }
    leaf source-address {
        type inet:ip-address-no-zone;
        mandatory true;
        description
            "The source address for the notifications.";
    }
}

grouping receiver-info {
    description
        "Defines where and how to get notifications for a configured
        subscriptions to one or more targeted recipient. This includes
        specifying the destination addressing as well as a transport
        protocol acceptable to the receiver.";
    container receivers {
        description
            "Set of receivers in a subscription.";
        list receiver {
            key "address port";
            min-elements 1;
            description
                "A single host or multipoint address intended as a target
                for the notifications for a subscription.";
            leaf address {
                type inet:host;
                mandatory true;
                description
                    "Specifies the address for the traffic to reach a remote
                    host. One of the following must be specified: an ipv4
                    address, an ipv6 address, or a host name.";
            }
            leaf port {
                type inet:port-number;
                mandatory true;
                description
                    "This leaf specifies the port number to use for messages
                    destined for a receiver.";
            }
        }
    }
}
```

```
    leaf protocol {
        type transport-protocol;
        default "netconf";
        description
            "This leaf specifies the transport protocol used
            to deliver messages destined for the receiver.  Each
            protocol may use the address and port information
            differently as applicable.";
    }
}
}
}

grouping error-identifier {
    description
        "A code passed back within an RPC response to describe why the RFC
        has failed, or within a state change notification to describe why
        the change has occurred.";
    leaf error-id {
        type subscription-errors;
        mandatory true;
        description
            "Identifies the subscription error condition.";
    }
}

grouping error-hints {
    description
        "Objects passed back within an RPC response to describe why the
        RFC has failed, or within a state change notification to
        describe why the change has occurred.";
    leaf filter-failure {
        type string;
        description
            "Information describing where and/or why a provided filter was
            unsupported for a subscription.";
    }
}

grouping subscription-response-with-hints {
    description
        "Defines the output for the establish-subscription and
        modify-subscription RPCs.";
    leaf subscription-result {
        type subscription-result;
        mandatory true;
        description
            "Indicates whether subscription is operational, or if a problem
```

```
        was encountered.";
    }
    choice result {
        description
            "Depending on the subscription result, different data is
            returned.";
        case no-success {
            description
                "This case applies when a subscription request was not
                successful and no subscription was created (or modified) as a
                result. In this case, information MAY be returned that
                indicates suggested parameter settings that would have a
                high likelihood of succeeding in a subsequent establish-
                subscription or modify-subscription request.";
            uses error-hints;
        }
    }
}

/*
 * RPCs
 */

rpc establish-subscription {
    description
        "This RPC allows a subscriber to create (and possibly negotiate)
        a subscription on its own behalf. If successful, the
        subscription remains in effect for the duration of the
        subscriber's association with the publisher, or until the
        subscription is terminated. In case an error (as indicated by
        subscription-result) is returned, the subscription is not
        created. In that case, the RPC output MAY include suggested
        parameter settings that would have a high likelihood of
        succeeding in a subsequent establish-subscription request.";
    input {
        uses subscription-policy;
    }
    output {
        uses subscription-response-with-hints {
            augment "result" {
                description
                    "Allows information to be passed back as part of a
                    successful subscription establishment.";
                case success {
                    description
                        "This case is used when the subscription request was
                        successful.";
                    leaf identifier {
```

```

        type subscription-id;
        mandatory true;
        description
            "Identifier used for this subscription.";
    }
}
}
augment "result/no-success" {
    description
        "Contains establish RPC specific objects which can be
        returned as hints for future attempts.";
    leaf replay-start-time-hint {
        type yang:date-and-time;
        description
            "If a replay has been requested, but the requested replay
            time cannot be honored, this may provide a hint at an
            alternate time which may be supportable.";
    }
}
}
}
}

rpc modify-subscription {
    description
        "This RPC allows a subscriber to modify a subscription that was
        previously created using establish-subscription.  If successful,
        the changed subscription remains in effect for the duration of
        the subscriber's association with the publisher, or until the
        subscription is again modified or terminated.  In case an error
        is returned (as indicated by subscription-result), the
        subscription is not modified and the original subscription
        parameters remain in effect.  In that case, the rpc error
        response MAY include suggested parameter hints that would have
        a high likelihood of succeeding in a subsequent
        modify-subscription request.";
    input {
        leaf identifier {
            type subscription-id;
            description
                "Identifier to use for this subscription.";
        }
        uses subscription-policy-modifiable;
    }
    output {
        uses subscription-response-with-hints;
    }
}
}

```

```
rpc delete-subscription {
  description
    "This RPC allows a subscriber to delete a subscription that
    was previously created from by that same subscriber using the
    establish-subscription RPC.";
  input {
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "Identifier of the subscription that is to be deleted.
        Only subscriptions that were created using
        establish-subscription can be deleted via this RPC.";
    }
  }
  output {
    leaf subscription-result {
      type subscription-result;
      mandatory true;
      description
        "Indicates whether subscription has been deleted, or if a
        problem was encountered.";
    }
  }
}

rpc kill-subscription {
  description
    "This RPC allows an operator to delete a dynamic subscription
    without restrictions on the originating subscriber or underlying
    transport session.";
  input {
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "Identifier of the subscription that is to be deleted. Only
        subscriptions that were created using establish-subscription
        can be deleted via this RPC.";
    }
  }
  output {
    leaf subscription-result {
      type subscription-result;
      mandatory true;
      description
        "Indicates whether subscription has been killed, or if a
        problem was encountered.";
    }
  }
}
```

```
    }  
  }  
}  
  
/*  
 * NOTIFICATIONS  
 */  
  
notification replay-complete {  
  sn:subscription-state-notif;  
  description  
    "This notification is sent to indicate that all of the replay  
    notifications have been sent. It must not be sent for any other  
    reason.";  
  leaf identifier {  
    type subscription-id;  
    mandatory true;  
    description  
      "This references the affected subscription.";  
  }  
}  
  
notification notification-complete {  
  sn:subscription-state-notif;  
  description  
    "This notification is sent to indicate that a subscription has  
    finished passing events.";  
  leaf identifier {  
    type subscription-id;  
    mandatory true;  
    description  
      "This references the affected subscription.";  
  }  
}  
  
notification subscription-started {  
  sn:subscription-state-notif;  
  description  
    "This notification indicates that a subscription has started and  
    notifications are beginning to be sent. This notification shall  
    only be sent to receivers of a subscription; it does not  
    constitute a general-purpose notification.";  
  leaf identifier {  
    type subscription-id;  
    mandatory true;  
    description  
      "This references the affected subscription.";  
  }  
}
```



```
    uses subscription-policy;
  }

notification subscription-resumed {
  sn:subscription-state-notif;
  description
    "This notification indicates that a subscription that had
    previously been suspended has resumed. Notifications will once
    again be sent.";
  leaf identifier {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
}

notification subscription-modified {
  sn:subscription-state-notif;
  description
    "This notification indicates that a subscription has been
    modified. Notifications sent from this point on will conform to
    the modified terms of the subscription. For completeness, this
    notification includes both modified and non-modified aspects of
    a subscription ";
  leaf identifier {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
  uses subscription-policy;
}

notification subscription-terminated {
  sn:subscription-state-notif;
  description
    "This notification indicates that a subscription has been
    terminated.";
  leaf identifier {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
  uses error-identifier;
  uses error-hints;
}
```

```
notification subscription-suspended {
  sn:subscription-state-notif;
  description
    "This notification indicates that a suspension of the
    subscription by the publisher has occurred. No further
    notifications will be sent until the subscription resumes.
    This notification shall only be sent to receivers of a
    subscription; it does not constitute a general-purpose
    notification.";
  leaf identifier {
    type subscription-id;
    mandatory true;
    description
      "This references the affected subscription.";
  }
  uses error-identifier;
  uses error-hints;
}

/*
 * DATA NODES
 */

container streams {
  config false;
  description
    "This container contains a leaf list of built-in
    streams that are provided by the system.";
  leaf-list stream {
    type stream;
    description
      "Identifies the built-in streams that are supported by the
      system. Built-in streams are associated with their own
      identities, each of which carries a special semantics.
      In case configurable custom streams are supported,
      as indicated by the custom-stream identity, the configuration
      of those custom streams is provided separately.";
  }
}

container filters {
  description
    "This container contains a list of configurable filters
    that can be applied to subscriptions. This facilitates
    the reuse of complex filters once defined.";
  list filter {
    key "identifier";
    description
```

```
        "A list of configurable filters that can be applied to
        subscriptions.";
    leaf identifier {
        type filter-id;
        description
            "An identifier to differentiate between filters.";
    }
    uses base-filter;
}
}
container subscription-config {
    if-feature "configured-subscriptions";
    description
        "Contains the list of subscriptions that are configured,
        as opposed to established via RPC or other means.";
    list subscription {
        key "identifier";
        description
            "Content of a subscription.";
        leaf identifier {
            type subscription-id;
            description
                "Identifier to use for this subscription.";
        }
        uses subscription-policy-non-modifiable;
        uses subscription-policy-modifiable;
        uses receiver-info {
            if-feature "configured-subscriptions";
        }
        uses notification-origin-info {
            if-feature "configured-subscriptions";
        }
    }
}
}
container subscriptions {
    config false;
    description
        "Contains the list of currently active subscriptions, i.e.
        subscriptions that are currently in effect, used for subscription
        management and monitoring purposes. This includes subscriptions
        that have been setup via RPC primitives as well as subscriptions
        that have been established via configuration.";
    list subscription {
        key "identifier";
        config false;
        description
            "Content of a subscription. Subscriptions can be created using
            a control channel or RPC, or be established through
```

```
        configuration.";
    leaf identifier {
        type subscription-id;
        description
            "Identifier of this subscription.";
    }
    leaf configured-subscription {
        if-feature "configured-subscriptions";
        type empty;
        description
            "The presence of this leaf indicates that the subscription
            originated from configuration, not through a control channel
            or RPC.";
    }
    uses subscription-policy;
    uses notification-origin-info {
        if-feature "configured-subscriptions";
    }
    uses receiver-info {
        refine receivers/receiver {
            min-elements "1";
        }
        augment receivers/receiver {
            description
                "include operational data for receivers.";
            leaf pushed-notifications {
                type yang:counter64;
                description
                    "Operational data which provides the number of update
                    notifications pushed to a receiver.";
            }
            leaf excluded-notifications {
                type yang:counter64;
                description
                    "Operational data which provides the number of non-
                    datastore update notifications explicitly removed via
                    filtering so that they are not sent to a receiver.";
            }
            leaf status {
                type subscription-status;
                mandatory true;
                description
                    "The status of the subscription.";
            }
        }
    }
}
```

```
}  
<CODE ENDS>
```

10. Considerations

10.1. Implementation Considerations

For a deployment including both configured and dynamic subscriptions, split subscription identifiers into static and dynamic halves. That way there should not be collisions if the configured subscriptions attempt to set a subscription-id which might have already been dynamically allocated. The lower half should be used for configured subscriptions and upper half for dynamic.

The <notification> elements are never sent before the transport layer, including capabilities exchange, has been established.

It is left to an implementation to determine when to transition between active and suspended subscription states. However if a subscription is unable to marshal all intended updates into a transmittable message in multiple successive intervals, the subscription should be suspended with the reason "unsupportable-volume".

10.2. Security Considerations

For dynamic subscriptions the publisher must authenticate and authorize all RPC requests.

Subscriptions could overload a publisher's CPU. For this reason, the publisher must have the ability to decline a dynamic subscription request, and provide the appropriate RPC error response to a subscriber should the proposed subscription overly deplete the publisher's resources.

A publisher needs to be able to suspend an existing dynamic or configured subscription based on capacity constraints. When this occur, the subscription status must be updated accordingly and the receivers notified with subscription state notifications.

If a malicious or buggy subscriber sends an unexpectedly large number of RPCs, this may use up system resources. In such a situation, subscription interactions may be terminated by terminating the transport session.

For both configured and dynamic subscriptions the publisher must authenticate and authorize a receiver via some transport level mechanism before sending any updates.

A secure transport is highly recommended and the publisher must ensure that the user has sufficient authorization to perform the function they are requesting against the specific subset of content involved.

A publisher **MUST NOT** include any content in a notification for which the user has not been authorized.

With configured subscriptions, one or more publishers could be used to overwhelm a receiver. No push updates should be sent to any receiver which doesn't even support subscriptions. Subscribers that do not want pushed data need only terminate or refuse any transport sessions from the publisher.

The NETCONF Authorization Control Model [RFC6536bis] **SHOULD** be used to control and restrict authorization of subscription configuration. This control models permits specifying per-user permissions to receive events from specific streams.

Where NACM is available, the NACM "very-secure" tag should be placed on the <kill-subscription> RPC so that only administrators can access.

One subscription id can be used for two or more receivers of the same configured subscription. But due to the possibility of different access control permissions per receiver, it should not be assumed that each receiver is getting identical updates.

11. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Tim Jenkins, Balazs Lengyel, Sharon Chisholm, Hector Trevino, Susan Hares, Kent Watsen, Michael Scharf, and Guangying Zheng.

12. References

12.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<http://www.rfc-editor.org/info/rfc3339>>.

- [RFC6536bis]
Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-01 (work in progress), March 2017.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", November 1999, <<http://www.w3.org/TR/1999/REC-xpath-19991116>>.

12.2. Informative References

- [I-D.ietf-netconf-event-notif]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Nilsen-Nygaard, E., Tripathy, A., Chisholm, S., and H. Trevino, "NETCONF support for event notifications", August 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-event-notifications/>>.
- [I-D.ietf-netconf-restconf-notif]
Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and HTTP transport for event notifications", August 2016, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf-notif/>>.
- [I-D.ietf-netconf-yang-push]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", June 2017, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>>.
- [I-D.voit-notifications2]
Voit, Eric., Clemm, Alexander., Bierman, A., and T. Jenkins, "YANG Notification Headers and Bundles", July 2017, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-notification-messages>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<http://www.rfc-editor.org/info/rfc5277>>.

- [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>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.
- [RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <<http://www.rfc-editor.org/info/rfc7923>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

Appendix A. Relationships to other drafts

There are other related drafts which are progressing in the NETCONF WG. This section details the relationship of this draft to those others.

A.1. ietf-netconf-netconf-event-notif

The [I-D.ietf-netconf-event-notif] draft augments this subscribed-notifications specification by defining NETCONF transport specifics. Included are:

- o bindings for RPC communications and Event Notifications over NETCONF.
- o encoded examples

A.2. ietf-netconf-restconf-notif

The [I-D.ietf-netconf-restconf-notif] draft augments this subscribed-notifications specification by defining transport specific guidance where some form of HTTP is used underneath. Included are:

- o bindings for RPC communications over RESTCONF
- o bindings for Event Notifications over HTTP2 and HTTP1.1
- o encoded examples

- o end-to-end deployment guidance for Call Home and TLS Heartbeat

A.3. ietf-netconf-yang-push

The draft [I-D.ietf-netconf-yang-push] builds upon this subscribed-notifications specification in order to allow a Publisher to stream YANG datastore objects. In this case, the application of either an on-change or periodic triggers upon a YANG Datastore replace the system generated events within this document.

If you wish to subscribe to a YANG datastore rather than a existing event stream on a publisher, please refer to this specification.

A.4. voit-notifications2

The draft [I-D.voit-notifications2] is not required to implement this subscribed-notifications specification. Instead it defines data plane notification elements which improve the delivered experience. The following capabilities are specified:

- o Defines common encapsulation headers objects to support functionality such as event severity, message signing, message loss discovery, message de-duplication, originating process identification.
- o Defines how to bundle multiple event records into a single notification message.

These are the enhanced capabilities alluded to in the Event (Data Plane) Notification section above. This draft is not yet adopted by the NETCONF WG.

Appendix B. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

Issue #6: Data plane notifications and layered headers

How to allow for seamless integration with non-standard encodings and transports (like GPB/GRPC). Specify requirements encoding and transport must meet, provide examples.

Appendix C. Changes between revisions

(To be removed by RFC editor prior to publication)

v02 - v03

- o RPCs and Notification support is identified by the Notification 2.0 capability.
- o Updates to filtering identities and text
- o New error type for unsupportable volume of updates
- o Text tweaks.

v01 - v02

- o Subscription status moved under receiver.

v00 - v01

- o Security considerations updated
- o Intro rewrite, as well as scattered text changes
- o Added Appendix A, to help match this to related drafts in progress
- o Updated filtering definitions, and filter types in yang file, and moved to identities for filter types
- o Added Syslog as a stream
- o HTTP2 moved in from YANG-Push as a transport option
- o Replay made an optional feature for events. Won't apply to datastores
- o Enabled notification timestamp to have different formats.
- o Two error codes added.

v01 5277bis - v00 subscribed notifications

- o Kill subscription RPC added.
- o Renamed from 5277bis to Subscribed Notifications.
- o Changed the notification capabilities version from 1.1 to 2.0.
- o Extracted create-subscription and other elements of RFC5277.
- o Error conditions added, and made specific in return codes.
- o Simplified yang model structure for removal of 'basic' grouping.

- o Added a grouping for items which cannot be statically configured.
- o Operational counters per receiver.
- o Subscription-id and filter-id renamed to identifier
- o Section for replay added. Replay now cannot be configured.
- o Control plane notification renamed to subscription state notification
- o Source address: Source-vrf changed to string, default address option added
- o In yang model: 'info' changed to 'policy'
- o Scattered text clarifications

v00 - v01 of 5277bis

- o YANG Model changes. New groupings for subscription info to allow restriction of what is changeable via RPC. Removed notifications for adding and removing receivers of configured subscriptions.
- o Expanded/renamed definitions from event server to publisher, and client to subscriber as applicable. Updated the definitions to include and expand on RFC 5277.
- o Removal of redundancy with other drafts
- o Many other clean-ups of wording and terminology

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Alberto Gonzalez Prieto
VMWare

Email: agonzalezpri@vmware.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 15, 2017

K. Watsen
Juniper Networks
G. Wu
Cisco Systems
June 13, 2017

TLS Client and Server Models
draft-ietf-netconf-tls-client-server-03

Abstract

This document defines three YANG modules: the first defines groupings for a generic TLS client, the second defines groupings for a generic TLS server, and the third defines common identities and groupings used by both the client and the server. It is intended that these groupings will be used by applications using the TLS protocol.

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

This document contains references to other drafts in progress, both in the Normative References section, as well as in body text throughout. Please update the following references to reflect their final RFC assignments:

- o I-D.ietf-netconf-keystore

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- o "XXXX" --> the assigned RFC value for this draft
- o "YYYY" --> the assigned RFC value for I-D.ietf-netconf-keystore

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- o "2017-06-13" --> the publication date of this draft

The following Appendix section is to be removed prior to publication:

- o Appendix A. Change Log

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 December 15, 2017.

Copyright Notice

Copyright (c) 2017 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	3
1.1. Terminology	3
1.2. Tree Diagrams	3
2. The TLS Client Model	4
2.1. Tree Diagram	4
2.2. Example Usage	5
2.3. YANG Model	5
3. The TLS Server Model	8
3.1. Tree Diagram	9
3.2. Example Usage	9
3.3. YANG Model	10
4. The TLS Common Model	13
4.1. Tree Diagram	13
4.2. Example Usage	13

4.3. YANG Model	14
5. Security Considerations	21
6. IANA Considerations	22
6.1. The IETF XML Registry	22
6.2. The YANG Module Names Registry	22
7. Acknowledgements	23
8. References	23
8.1. Normative References	23
8.2. Informative References	24
Appendix A. Change Log	26
A.1. server-model-09 to 00	26
A.2. 00 to 01	26
A.3. 01 to 02	26
A.4. 02 to 03	26
Authors' Addresses	26

1. Introduction

This document defines three YANG [RFC7950] modules: the first defines a grouping for a generic TLS client, the second defines a grouping for a generic TLS server, and the third defines identities and groupings common to both the client and the server (TLS is defined in [RFC5246]). It is intended that these groupings will be used by applications using the TLS protocol. For instance, these groupings could be used to help define the data model for an HTTPS [RFC2818] server or a NETCONF over TLS [RFC7589] based server.

The client and server YANG modules in this document each define one grouping, which is focused on just TLS-specific configuration, and specifically avoids any transport-level configuration, such as what ports to listen-on or connect-to. This enables applications the opportunity to define their own strategy for how the underlying TCP connection is established. For instance, applications supporting NETCONF Call Home [RFC8071] could use the grouping for the TLS parts it provides, while adding data nodes for the TCP-level call-home configuration.

1.1. Terminology

The keywords "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].

1.2. Tree Diagrams

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. The TLS Client Model

The TLS client model presented in this section contains one YANG grouping, to just configure the TLS client omitting, for instance, any configuration for which IP address or port the client should connect to.

This grouping references data nodes defined by the keystore model [I-D.ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate which trusted CA certificate a client should use to authenticate the server's certificate.

2.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-tls-client module. Please see Section 1.2 for tree diagram notation.


```

module: ietf-tls-client
groupings:
  tls-client-grouping
    +----- server-auth
    |   +----- trusted-ca-certs?
    |   |           -> /ks:keystore/trusted-certificates/name
    |   +----- trusted-server-certs?
    |   |           -> /ks:keystore/trusted-certificates/name
    +----- client-auth
    |   +----- (auth-type)?
    |   |   +---:(certificate)
    |   |   +----- certificate?   leafref
    +----- hello-params {tls-client-hello-params-config}?
    |   +----- tls-versions
    |   |   +----- tls-version*   identityref
    +----- cipher-suites
    |   +----- cipher-suite*   identityref

```

2.2. Example Usage

This section shows how it would appear if the `tls-client-grouping` were populated with some data. This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

```

<!-- hypothetical example, as groupings don't have instance data -->
<tls-client xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-client">

  <!-- which certificates will this client trust -->
  <server-auth>
    <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
    <trusted-server-certs>explicitly-trusted-client-certs</trusted-server-certs>
  </server-auth>

  <!-- how this client will authenticate itself to the server -->
  <client-auth>
    <certificate>builtin-idevid-cert</certificate>
  </client-auth>

</tls-client>

```

2.3. YANG Model

This YANG module has a normative references to [RFC6991] and [I-D.ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-tls-client@2017-06-13.yang"
```

```
module ietf-tls-client {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-client";
  prefix "tlsc";

  import ietf-tls-common {
    prefix tlscom;
    revision-date 2017-06-13; // stable grouping definitions
    reference
      "RFC XXXX: TLS Client and Server Models";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC YYYY: Keystore Model";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>

    Author:     Kent Watsen
                 <mailto:kwatsen@juniper.net>";

  description
    "This module defines a reusable grouping for a TLS client that
    can be used as a basis for specific TLS client instances.

    Copyright (c) 2014 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD
    License set forth in Section 4.c of the IETF Trust's
    Legal Provisions Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX; see
    the RFC itself for full legal notices.";

  revision "2017-06-13" {
```

```
    description
      "Initial version";
    reference
      "RFC XXXX: TLS Client and Server Models";
  }

  feature tls-client-hello-params-config {
    description
      "TLS hello message parameters are configurable on a TLS
      client.";
  }

  grouping tls-client-grouping {
    description
      "A reusable grouping for configuring a TLS client without
      any consideration for how an underlying TCP session is
      established.";

    container server-auth {
      must 'trusted-ca-certs or trusted-server-certs';
      description
        "Trusted server identities.";
      leaf trusted-ca-certs {
        type leafref {
          path "/ks:keystore/ks:trusted-certificates/ks:name";
        }
        description
          "A reference to a list of certificate authority (CA)
          certificates used by the TLS client to authenticate
          TLS server certificates. A server certificate is
          authenticated if it has a valid chain of trust to
          a configured trusted CA certificate.";
      }

      leaf trusted-server-certs {
        type leafref {
          path "/ks:keystore/ks:trusted-certificates/ks:name";
        }
        description
          "A reference to a list of server certificates used by
          the TLS client to authenticate TLS server certificates.
          A server certificate is authenticated if it is an
          exact match to a configured trusted server certificate.";
      }
    }
  }

  container client-auth {
    description
```

```
        "The credentials used by the client to authenticate to
        the TLS server.";

    choice auth-type {
        description
            "The authentication type.";
        leaf certificate {
            type leafref {
                path "/ks:keystore/ks:keys/ks:key/ks:certificates"
                    + "/ks:certificate/ks:name";
            }
            description
                "A certificates to be used for user authentication.";
        }
    }
}

container hello-params {
    if-feature tls-client-hello-params-config;
    uses tlscom:hello-params-grouping;
    description
        "Configurable parameters for the TLS hello message.";
}

} // end tls-client-grouping

}
```

<CODE ENDS>

3. The TLS Server Model

The TLS server model presented in this section contains one YANG grouping, for just the TLS-level configuration omitting, for instance, configuration for which ports to open to listen for connections on.

This grouping references data nodes defined by the keystore model [I-D.ietf-netconf-keystore]. For instance, a reference to the keystore model is made to indicate which certificate a server should present.

3.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-tls-server module. Please see Section 1.2 for tree diagram notation.

```

module: ietf-tls-server
  groupings:
    tls-server-grouping
      +---- certificates
      |   +---- certificate* [name]
      |   |   +---- name? leafref
      |   +---- client-auth
      |   |   +---- trusted-ca-certs?
      |   |   |   -> /ks:keystore/trusted-certificates/name
      |   |   +---- trusted-client-certs?
      |   |   |   -> /ks:keystore/trusted-certificates/name
      +---- hello-params {tls-server-hello-params-config}?
      +---- tls-versions
      |   +---- tls-version* identityref
      +---- cipher-suites
      |   +---- cipher-suite* identityref

```

3.2. Example Usage

This section shows how it would appear if the tls-server-grouping were populated with some data. This example is consistent with the examples presented in Section 2.2 of [I-D.ietf-netconf-keystore].

<!-- hypothetical example, groupings don't have instance data -->

```

<tls-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">
  <certificates>
    <certificate>
      <name>tls-ec-cert</name>
    </certificate>
  </certificates>
  <client-auth>
    <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
    <trusted-client-certs>explicitly-trusted-client-certs</trusted-client-certs>
  </client-auth>
</tls-server>

```

3.3. YANG Model

This YANG module has a normative references to [RFC6991], and [I-D.ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-tls-server@2017-06-13.yang"

module ietf-tls-server {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-server";
  prefix "tlss";

  import ietf-tls-common {
    prefix tlscom;
    revision-date 2017-06-13; // stable grouping definitions
    reference
      "RFC XXXX: TLS Client and Server Models";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC YYYY: Keystore Model";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>

    Author:     Kent Watsen
                <mailto:kwatsen@juniper.net>";

  description
    "This module defines a reusable grouping for a TLS server that
    can be used as a basis for specific TLS server instances.

    Copyright (c) 2014 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD
```

License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

```
revision "2017-06-13" {
  description
    "Initial version";
  reference
    "RFC XXXX: TLS Client and Server Models";
}

feature tls-server-hello-params-config {
  description
    "TLS hello message parameters are configurable on a TLS
    server.";
}

// grouping
grouping tls-server-grouping {
  description
    "A reusable grouping for configuring a TLS server without
    any consideration for how underlying TCP sessions are
    established.";
  container certificates {
    description
      "The list of certificates the TLS server will present when
      establishing a TLS connection in its Certificate message,
      as defined in Section 7.4.2 in RFC 5246.";
    reference
      "RFC 5246:
      The Transport Layer Security (TLS) Protocol Version 1.2";
    list certificate {
      key name;
      min-elements 1;
      description
        "An unordered list of certificates the TLS server can pick
        from when sending its Server Certificate message.";
      reference
        "RFC 5246: The TLS Protocol, Section 7.4.2";
      leaf name {
        type leafref {
          path "/ks:keystore/ks:keys/ks:key/ks:certificates/"
            + "ks:certificate/ks:name";
        }
      }
      description

```

```
        "The name of the certificate in the keystore.";
    }
}

container client-auth {
  description
    "A reference to a list of trusted certificate authority (CA)
    certificates and a reference to a list of trusted client
    certificates.";
  leaf trusted-ca-certs {
    type leafref {
      path "/ks:keystore/ks:trusted-certificates/ks:name";
    }
    description
      "A reference to a list of certificate authority (CA)
      certificates used by the TLS server to authenticate
      TLS client certificates.";
  }
  leaf trusted-client-certs {
    type leafref {
      path "/ks:keystore/ks:trusted-certificates/ks:name";
    }
    description
      "A reference to a list of client certificates used by
      the TLS server to authenticate TLS client certificates.
      A clients certificate is authenticated if it is an
      exact match to a configured trusted client certificate.";
  }
}

container hello-params {
  if-feature tls-server-hello-params-config;
  uses tlscom:hello-params-grouping;
  description
    "Configurable parameters for the TLS hello message.";
}

} // end tls-server-grouping
}
```

<CODE ENDS>

4. The TLS Common Model

The TLS common model presented in this section contains identities and groupings common to both TLS clients and TLS servers. The hello-params-grouping can be used to configure the list of TLS algorithms permitted by the TLS client or TLS server. The lists of algorithms are ordered such that, if multiple algorithms are permitted by the client, the algorithm that appears first in its list that is also permitted by the server is used for the TLS transport layer connection. The ability to restrict the the algorithms allowed is provided in this grouping for TLS clients and TLS servers that are capable of doing so and may serve to make TLS clients and TLS servers compliant with security policies.

Features are defined for algorithms that are OPTIONAL or are not widely supported by popular implementations. Note that the list of algorithms is not exhaustive.

4.1. Tree Diagram

The following tree diagram presents the data model for the grouping defined in the ietf-tls-common module. Please see Section 1.2 for tree diagram notation.

```
module: ietf-tls-common
  groupings:
    hello-params-grouping
      +---- tls-versions
      |   +---- tls-version*   identityref
      +---- cipher-suites
      |   +---- cipher-suite*   identityref
```

4.2. Example Usage

This section shows how it would appear if the transport-params-grouping were populated with some data.

```
<!-- hypothetical example, as groupings don't have instance data -->
<hello-params xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-common">

  <tls-versions>
    <tls-version>tls-1.2</tls-version>
  </tls-versions>
  <cipher-suites>
    <cipher-suite>ecdhe-rsa-with-3des-edc-cbc-sha</cipher-suite>
    <cipher-suite>dhe-rsa-with-aes-128-cbc-sha</cipher-suite>
    <cipher-suite>rsa-with-aes-128-cbc-sha</cipher-suite>
    <cipher-suite>rsa-with-3des-edc-cbc-sha</cipher-suite>
  </cipher-suites>

</hello-params>
```

4.3. YANG Model

This YANG module has a normative references to [RFC4492], [RFC5246], [RFC5288], and [RFC5289].

```
<CODE BEGINS> file "ietf-tls-common@2017-06-13.yang"

module ietf-tls-common {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-common";
  prefix "tlscom";

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/netconf/>
    WG List:  <mailto:netconf@ietf.org>

    Author:   Kent Watsen
              <mailto:kwatsen@juniper.net>

    Author:   Gary Wu
              <mailto:garywu@cisco.com>";

  description
    "This module defines a common features, identities, and groupings
    for Transport Layer Security (TLS).

    Copyright (c) 2017 IETF Trust and the persons identified as
```

authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

```
revision "2017-06-13" {
  description
    "Initial version";
  reference
    "RFC XXXX: TLS Client and Server Models";
}

// features
feature tls-ecc {
  description
    "Elliptic Curve Cryptography (ECC) is supported for TLS.";
  reference
    "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites
      for Transport Layer Security (TLS)";
}

feature tls-dhe {
  description
    "Ephemeral Diffie-Hellman key exchange is supported for TLS.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
}

feature tls-3des {
  description
    "The Triple-DES block cipher is supported for TLS.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
}

feature tls-gcm {
  description
    "The Galois/Counter Mode authenticated encryption mode is
      supported for TLS.";
```

```
    reference
      "RFC 5288: AES Galois Counter Mode (GCM) Cipher Suites for TLS";
  }

  feature tls-sha2 {
    description
      "The SHA2 family of cryptographic hash functions is supported
      for TLS.";
    reference
      "FIPS PUB 180-4: Secure Hash Standard (SHS)";
  }

  // identities
  identity tls-version-base {
    description
      "Base identity used to identify TLS protocol versions.";
  }

  identity tls-1.2 {
    base tls-version-base;
    description
      "TLS protocol version 1.2.";
    reference
      "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
  }

  identity cipher-suite-base {
    description
      "Base identity used to identify TLS cipher suites.";
  }

  identity rsa-with-aes-128-cbc-sha {
    base cipher-suite-base;
    description
      "Cipher suite TLS_RSA_WITH_AES_128_CBC_SHA.";
    reference
      "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
  }

  identity rsa-with-aes-256-cbc-sha {
    base cipher-suite-base;
    description
      "Cipher suite TLS_RSA_WITH_AES_256_CBC_SHA.";
    reference
      "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
  }
```

```
}

identity rsa-with-aes-128-cbc-sha256 {
  base cipher-suite-base;
  if-feature tls-sha2;
  description
    "Cipher suite TLS_RSA_WITH_AES_128_CBC_SHA256.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
}

identity rsa-with-aes-256-cbc-sha256 {
  base cipher-suite-base;
  if-feature tls-sha2;
  description
    "Cipher suite TLS_RSA_WITH_AES_256_CBC_SHA256.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
}

identity dhe-rsa-with-aes-128-cbc-sha {
  base cipher-suite-base;
  if-feature tls-dhe;
  description
    "Cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
}

identity dhe-rsa-with-aes-256-cbc-sha {
  base cipher-suite-base;
  if-feature tls-dhe;
  description
    "Cipher suite TLS_DHE_RSA_WITH_AES_256_CBC_SHA.";
  reference
    "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
}

identity dhe-rsa-with-aes-128-cbc-sha256 {
  base cipher-suite-base;
  if-feature "tls-dhe and tls-sha2";
  description
    "Cipher suite TLS_DHE_RSA_WITH_AES_128_CBC_SHA256.";
  reference
```

```
    "RFC 5246: The Transport Layer Security (TLS) Protocol
      Version 1.2";
  }

  identity dhe-rsa-with-aes-256-cbc-sha256 {
    base cipher-suite-base;
    if-feature "tls-dhe and tls-sha2";
    description
      "Cipher suite TLS_DHE_RSA_WITH_AES_256_CBC_SHA256.";
    reference
      "RFC 5246: The Transport Layer Security (TLS) Protocol
        Version 1.2";
  }

  identity ecdhe-ecdsa-with-aes-128-cbc-sha256 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-sha2";
    description
      "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256.";
    reference
      "RFC 5289: TLS Elliptic Curve Cipher Suites with
        SHA-256/384 and AES Galois Counter Mode (GCM)";
  }

  identity ecdhe-ecdsa-with-aes-256-cbc-sha384 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-sha2";
    description
      "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384.";
    reference
      "RFC 5289: TLS Elliptic Curve Cipher Suites with
        SHA-256/384 and AES Galois Counter Mode (GCM)";
  }

  identity ecdhe-rsa-with-aes-128-cbc-sha256 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-sha2";
    description
      "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256.";
    reference
      "RFC 5289: TLS Elliptic Curve Cipher Suites with
        SHA-256/384 and AES Galois Counter Mode (GCM)";
  }

  identity ecdhe-rsa-with-aes-256-cbc-sha384 {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-sha2";
    description
```

```
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384.";
  reference
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
      SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-ecdsa-with-aes-128-gcm-sha256 {
  base cipher-suite-base;
  if-feature "tls-ecc and tls-gcm and tls-sha2";
  description
    "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256.";
  reference
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
      SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-ecdsa-with-aes-256-gcm-sha384 {
  base cipher-suite-base;
  if-feature "tls-ecc and tls-gcm and tls-sha2";
  description
    "Cipher suite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384.";
  reference
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
      SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-rsa-with-aes-128-gcm-sha256 {
  base cipher-suite-base;
  if-feature "tls-ecc and tls-gcm and tls-sha2";
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.";
  reference
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
      SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity ecdhe-rsa-with-aes-256-gcm-sha384 {
  base cipher-suite-base;
  if-feature "tls-ecc and tls-gcm and tls-sha2";
  description
    "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384.";
  reference
    "RFC 5289: TLS Elliptic Curve Cipher Suites with
      SHA-256/384 and AES Galois Counter Mode (GCM)";
}

identity rsa-with-3des-edc-cbc-sha {
  base cipher-suite-base;
```

```
    if-feature tls-3des;
    description
        "Cipher suite TLS_RSA_WITH_3DES_EDE_CBC_SHA.";
    reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
          Version 1.2";
}

identity ecdhe-rsa-with-3des-ede-cbc-sha {
    base cipher-suite-base;
    if-feature "tls-ecc and tls-3des";
    description
        "Cipher suite TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA.";
    reference
        "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites
          for Transport Layer Security (TLS)";
}

identity ecdhe-rsa-with-aes-128-cbc-sha {
    base cipher-suite-base;
    if-feature "tls-ecc";
    description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA.";
    reference
        "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites
          for Transport Layer Security (TLS)";
}

identity ecdhe-rsa-with-aes-256-cbc-sha {
    base cipher-suite-base;
    if-feature "tls-ecc";
    description
        "Cipher suite TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA.";
    reference
        "RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites
          for Transport Layer Security (TLS)";
}

// groupings
grouping hello-params-grouping {
    description
        "A reusable grouping for TLS hello message parameters.  For
         configurable parameters, a zero-element leaf-list indicates the
         system default configuration for that parameter.";
    reference
        "RFC 5246: The Transport Layer Security (TLS) Protocol
          Version 1.2";
    container tls-versions {
```



```
    description
      "Parameters regarding TLS versions.";
    leaf-list tls-version {
      type identityref {
        base tls-version-base;
      }
      description
        "Allowed TLS protocol versions.";
    }
  }
  container cipher-suites {
    description
      "Parameters regarding cipher suites.";
    leaf-list cipher-suite {
      type identityref {
        base cipher-suite-base;
      }
      ordered-by user;
      description
        "Cipher suites in order of descending preference.";
    }
  }
}
```

<CODE ENDS>

5. Security Considerations

The YANG module defined in this document is designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/: The entire data tree defined by this module is sensitive to write operations. For instance, the addition or removal of references to keys, certificates, trusted anchors, etc., can dramatically alter the implemented security policy. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated 'recovery session'.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

NONE

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

NONE

6. IANA Considerations

6.1. The IETF XML Registry

This document registers three URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tls-client
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tls-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tls-common
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

6.2. The YANG Module Names Registry

This document registers three YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the the following registrations are requested:

```
name:      ietf-tls-client
namespace: urn:ietf:params:xml:ns:yang:ietf-tls-client
prefix:    tlsc
reference:  RFC XXXX

name:      ietf-tls-server
namespace: urn:ietf:params:xml:ns:yang:ietf-tls-server
prefix:    tlss
reference:  RFC XXXX

name:      ietf-tls-common
namespace: urn:ietf:params:xml:ns:yang:ietf-tls-common
prefix:    tlss
reference:  RFC XXXX
```

7. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Benoit Claise, Mehmet Ersue, Balazs Kovacs, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Juergen Schoenwaelder, Phil Shafer, Sean Turner, and Bert Wijnen.

8. References

8.1. Normative References

- [I-D.ietf-netconf-keystore]
Watsen, K., "Keystore Model", draft-ietf-netconf-keystore-01 (work in progress), March 2017.
- [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>>.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <<http://www.rfc-editor.org/info/rfc4492>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.

- [RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, DOI 10.17487/RFC5288, August 2008, <<http://www.rfc-editor.org/info/rfc5288>>.
- [RFC5289] Rescorla, E., "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)", RFC 5289, DOI 10.17487/RFC5289, August 2008, <<http://www.rfc-editor.org/info/rfc5289>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <<http://www.rfc-editor.org/info/rfc7589>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.

8.2. Informative References

- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [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>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home",
RFC 8071, DOI 10.17487/RFC8071, February 2017,
<<http://www.rfc-editor.org/info/rfc8071>>.

Appendix A. Change Log

A.1. server-model-09 to 00

- o This draft was split out from draft-ietf-netconf-server-model-09.
- o Noted that '0.0.0.0' and ':::' might have special meanings.

A.2. 00 to 01

- o Renamed "keychain" to "keystore".

A.3. 01 to 02

- o Removed the groupings containing transport-level configuration. Now modules contain only the transport-independent groupings.
- o Filled in previously incomplete 'ietf-tls-client' module.
- o Added cipher suites for various algorithms into new 'ietf-tls-common' module.

A.4. 02 to 03

- o Added a 'must' statement to container 'server-auth' asserting that at least one of the various auth mechanisms must be specified.
- o Fixed description statement for leaf 'trusted-ca-certs'.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Systems

EMail: garywu@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: December 28, 2017

A. Clemm
Huawei
E. Voit
Cisco Systems
A. Gonzalez Prieto

A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Bierman
YumaWorks
B. Lengyel
Ericsson
June 26, 2017

Subscribing to YANG datastore push updates
draft-ietf-netconf-yang-push-07

Abstract

Providing rapid visibility into changes made on YANG configuration and operational objects enables new capabilities such as remote mirroring of configuration and operational state. Via the mechanism described in this document, subscriber applications may request a continuous, customized stream of updates from a YANG datastore.

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 December 28, 2017.

Copyright Notice

Copyright (c) 2017 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
2. Definitions and Acronyms	4
3. Solution Overview	5
3.1. Event Subscription Model	5
3.2. Negotiation of Subscription Policies	6
3.3. On-Change Considerations	6
3.4. Promise-Theory Considerations	8
3.5. Data Encodings	8
3.6. Datastore filters	9
3.7. Streaming updates	10
3.8. Subscription management	13
3.9. Receiver Authorization	14
3.10. On-change notifiable YANG objects	15
3.11. Other considerations	16
4. A YANG data model for management of datastore push subscriptions	18
4.1. Overview	18
4.2. Subscription configuration	23
4.3. YANG Notifications	25

4.4. YANG RPCs	26
5. YANG module	30
6. IANA Considerations	43
7. Security Considerations	44
8. Acknowledgments	44
9. References	44
9.1. Normative References	44
9.2. Informative References	45
Appendix A. Relationships to other drafts	46
A.1. ietf-netconf-subscribed-notifications	46
A.2. ietf-netconf-netconf-event-notif	46
A.3. ietf-netconf-restconf-notif	46
A.4. voit-notifications2	47
Appendix B. Technologies to be considered for future iterations	47
B.1. Proxy YANG Subscription when the Subscriber and Receiver are different	47
B.2. OpState and Filters	48
B.3. Splitting push updates	48
B.4. Potential Subscription Parameters	49
Appendix C. Issues that are currently being worked and resolved	49
Appendix D. Changes between revisions	49
Authors' Addresses	51

1. Introduction

Traditional approaches to remote visibility have been built on polling. With polling, data is periodically requested and retrieved by a client from a server to stay up-to-date. However, there are issues associated with polling-based management:

- o Polling incurs significant latency. This latency prohibits many application types.
- o Polling cycles may be missed, requests may be delayed or get lost, often when the network is under stress and the need for the data is the greatest.
- o Polling requests may undergo slight fluctuations, resulting in intervals of different lengths. The resulting data is difficult to calibrate and compare.
- o For applications that monitor for changes, many remote polling cycles place ultimately fruitless load on the network, devices, and applications.

A more effective alternative to polling is for an application to receive automatic and continuous updates from a targeted subset of a datastore. Accordingly, there is a need for a service that allows

applications to subscribe to updates from a YANG datastore and that enables the publisher to push and in effect stream those updates. The requirements for such a service have been documented in [RFC7923].

This document defines a corresponding solution that is built on top of "Custom Subscription to Event Notifications" [subscribe]. Supplementing that work are YANG data model augmentations, extended RPCs, and new datastore specific update notifications. Transport options for [subscribe] will work seamlessly with this solution.

2. Definitions and Acronyms

The terms below supplement those defined in [subscribe].

Data node: An instance of management information in a YANG datastore.

Data node update: A data item containing the current value/property of a Data node at the time the data node update was created.

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

Data subtree: An instantiated data node and the data nodes that are hierarchically contained within it.

Notification message: A transport encapsulated update record(s) and/or event notification(s) intended to be sent to a receiver.

Update notification message: A notification message that contains an update record.

Update record: A representation data node update(s) resulting from the application of a filter for a subscription. An update record will include the value/property of one or more data nodes at a point in time. It may contain the update type for each data node (e.g., add, change, delete). Also included may be metadata/headers such as a subscription-id.

Update trigger: A mechanism that determines when an update record needs to be generated.

YANG-Push: The subscription and push mechanism for YANG datastores that is specified in this document.

3. Solution Overview

This document specifies a solution for a push update subscription service. This solution supports the dynamic as well as configured subscriptions to information updates from YANG datastores. Subscriptions specify when update notification messages should be sent and what data to include in update records. YANG objects are subsequently pushed from the publisher to the receiver per the terms of the subscription.

3.1. Event Subscription Model

YANG-push subscriptions are defined using a data model that is itself defined in YANG. This model enhances the event subscription model defined in [subscribe] with capabilities that allow subscribers to subscribe to data node updates, specifically to specify the triggers when to generate update records as well as what to include in an update record. Key enhancements include:

- o Specification of selection filters which identify targeted YANG data nodes and/or subtrees within a datastore for which updates are to be pushed.
- o An encoding (using anydata) for the contents of periodic and on-change push updates.
- o Specification of update policies that specify the conditions that trigger the generation and pushing of new update records. There are two types of subscriptions, periodic and on-change.
 - * For periodic subscriptions, the trigger is specified by two parameters that define when updates are to be pushed. These parameters are the period interval with which to report updates, and an anchor time which can be used to calculate at which point in time updates need to be assembled and sent.
 - * For on-change subscriptions, a trigger occurs whenever a change in the subscribed information is detected. Included are additional parameters such as:
 - + Dampening period: In an on-change subscription, the first change that is detected results in an update to be sent immediately. However, sending successive updates whenever further changes are detected might result in quick exhaustion of resources in case of very rapid changes. In order to protect against that, a dampening period is used to specify the interval which must pass before successive update records for the same subscription are generated. The

dampening period collectively applies to the set of all data nodes of a single subscription. This means that on change of an object being subscribed to, an update record containing that object is created either immediately when no dampening period is already in effect, or at the end of a dampening period.

- + Change type: This parameter can be used to reduce the types of datastore changes for which updates are sent (e.g., you might only send when an object is created or deleted, but not when an object value changes).
- + No Synch on start: defines whether or not a complete push-update of all subscribed data will be sent at the beginning of a subscription. Such synchronization establishes the frame of reference for subsequent updates.

3.2. Negotiation of Subscription Policies

A dynamic subscription request SHOULD be declined based on publisher's assessment that it may be unable to provide update records that would meet the terms of the request. However a subscriber may quickly follow up with a new subscription request using different parameters.

Random guessing at different parameters by a subscriber is to be discouraged. Therefore, in order to minimize the number of subscription iterations between subscriber and publisher, dynamic subscriptions SHOULD support a simple negotiation between subscribers and publishers for subscription parameters. This negotiation is in the form of a no-success response to a failed establish or modify subscription request. The no-success message SHOULD include in the returned error response information that, when considered, increases the likelihood of success for subsequent requests. However, there are no guarantees that subsequent requests for this subscriber will in fact be accepted.

Such negotiation information returned from a publisher beyond that from [subscribe] includes hints at acceptable time intervals, size estimates for the number or objects which would be returned from a filter, and the names of targeted objects not found in the publisher's YANG tree.

3.3. On-Change Considerations

On-change subscriptions allow subscribers to subscribe to updates whenever changes to objects occur. As such, on-change subscriptions are particularly effective for data that changes infrequently, yet

that requires applications to be notified whenever a change does occur with minimal delay.

On-change subscriptions tend to be more difficult to implement than periodic subscriptions. Accordingly, on-change subscriptions may not be supported by all implementations or for every object.

Whether or not to accept or reject on-change subscription requests when the scope of the subscription contains objects for which on-change is not supported is up to the server implementation: A server MAY accept an on-change subscription even when the scope of the subscription contains objects for which on-change is not supported. In that case, updates are sent only for those objects within the scope that do support on-change updates whereas other objects are excluded from update records, whether or not their values actually change. In order for a client to determine whether objects support on-change subscriptions, objects are marked accordingly by a server. Accordingly, when subscribing, it is the responsibility of the client to ensure it is aware of which objects support on-change and which do not. For more on how objects are so marked, see Section 3.10.

Alternatively, a server MAY decide to simply reject an on-change subscription in case the scope of the subscription contains objects for which on-change is not supported. In case of a configured subscription, the subscription can be marked as suspended respectively inoperational.

To avoid flooding receivers with repeated updates for subscriptions containing fast-changing objects, or objects with oscillating values, an on-change subscription allows for the definition of a dampening period. Once an update record for a given object is generated, no other updates for this particular subscription will be created until the end of the dampening period. Values sent at the end of the dampening period are the current values of all changed objects which are current at the time the dampening period expires. Changed objects includes those which were deleted or newly created during that dampening period. If an object has returned to its original value (or even has been created and then deleted) during the dampening-period, the last change will still be sent. This will indicate churn is occurring on that object.

In cases where a client wants to have separate dampening periods for different objects, multiple subscriptions with different objects in subscription scope can be created.

On-change subscriptions can be refined to let users subscribe only to certain types of changes, for example, only to object creations and deletions, but not to modifications of object values.

3.4. Promise-Theory Considerations

A subscription to updates from a YANG datastore is intended to obviate the need for polling. However, in order to do so, it is of utmost importance that subscribers can rely on the subscription and have confidence that they will indeed receive the subscribed updates without having to worry updates being silently dropped. In other words, a subscription constitutes a promise on the side of the server to provide the receivers with updates per the terms of the subscription.

Now, there are many reasons why a server may at some point no longer be able to fulfill the terms of the subscription, even if the subscription had been entered into with good faith. For example, the volume of data objects may be larger than anticipated, the interval may prove too short to send full updates in rapid succession, or an internal problem may prevent updates from being collected. If for some reason the server of a subscription is not able to keep its promise, receivers **MUST** be notified immediately and reliably. The server **MUST** also update the state of the subscription to indicate that the subscription is in a detrimental state.

A server **SHOULD** reject a request for a subscription if it is unlikely that the server will be able fulfill the terms of the subscription. In such cases, it is preferable to have a client request another subscription that is less resource intensive (for example, a subscription with longer periodic update intervals), than to subsequently frustrate the receiver with 'frequent subscription suspensions.

3.5. Data Encodings

Subscribed data is encoded in either XML or JSON format. A publisher **MUST** support XML encoding and **MAY** support JSON encoding.

3.5.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been simply retrieved using a get operation and is encoded in the same way. XML encoding rules for data nodes are defined in [RFC7950]. JSON encoding rules are defined in [RFC7951].

3.5.2. On-Change Subscriptions

In an on-change subscription, updates need to indicate not only values of changed data nodes but also the types of changes that occurred since the last update. Therefore encoding rules for data in

on-change updates will follow YANG-patch operation as specified in [RFC8072]. The YANG-patch will describe what needs to be applied to the earlier state reported by the preceding update, to result in the now-current state. Note that contrary to [RFC8072], objects encapsulated are not restricted to configuration objects only.

3.6. Datastore filters

Subscription policy specifies both the filters and the datastores against which the filters will be applied. The result is the push of information necessary to remotely maintain an extract of publisher's datastore.

Only a single filter can be applied to a subscription at a time. The following selection filter types are included in the yang-push data model, and may be applied against a datastore:

- o subtree: A subtree filter identifies one or more subtrees. When specified, updates will only come from the data nodes of selected YANG subtree(s). The syntax and semantics correspond to that specified for [RFC6241] section 6.
- o xpath: An xpath filter is an XPath expression which may be meaningfully applied to a datastore. It is the results of this expression which will be pushed.

Filters are intended to be used as selectors that define which objects are within the scope of a subscription. Filters are not intended to be used to store objects based on their current value. Doing so would have a number of implications that would result in significant additional complexity. For example, withough extending encodings for on-change subscriptions, a receiver would not be able to distinguish cases in which an object is no longer included in an update because it was deleted, as opposed to its value simply no longer meeting the filter criteria. While it is possible to define extensions in the future that will support filtering based on values, this is not supported in this version of yang-push and a server MAY reject a subscription request that contains a filter for object values.

Xpath itself provides powerful filtering constructs, and care must be used in filter definition. As an example, consider an xpath filter with a boolean result; such a result will not provide an easily interpretable subset of a datastore. Beyond the boolean example, it is quite possible to define an xpath filter where results are easy for an application to mis-interpret. Consider an xpath filter which only passes a datastore object when interface=up. It is up to the

receiver to understand implications of the presence or absence of objects in each update.

It is not expected that implementations will support comprehensive filter syntax and boundless complexity. It will be up to implementations to describe what is viable, but the goal is to provide equivalent capabilities to what is available with a GET. Implementations **MUST** reject dynamic subscriptions or suspend configured subscriptions if they include filters which are unsupportable on a platform.

3.7. Streaming updates

Contrary to traditional data retrieval requests, datastore subscription enables an unbounded series of update records to be streamed over time. Two generic notifications for update records have been defined for this: "push-update" and "push-change-update".

A push-update notification defines a complete, filtered update of the datastore per the terms of a subscription. This type of notification is used for continuous updates of periodic subscriptions. A push-update notification can also be used for the on-change subscriptions in two cases. First it will be used as the initial push-update if there is a need to synchronize the receiver at the start of a new subscription. It also **MAY** be sent if the publisher later chooses to resynch an on-change subscription. The push-update record contains a data snippet that contains an instantiated subtree with the subscribed contents. The content of the update record is equivalent to the contents that would be obtained had the same data been explicitly retrieved using e.g., a NETCONF "get" operation, with the same filters applied.

A push-change-update notification is the most common type of update for on-change subscriptions. The update record in this case contains a data snippet that indicates the set of changes that data nodes have undergone since the last notification of YANG objects. In other words, this indicates which data nodes have been created, deleted, or have had changes to their values. In cases where multiple changes have occurred and the object has not been deleted, the object's most current value is reported. (In other words, for each object, only one change is reported, not its entire history. Doing so would defeat the purpose of the dampening period.)

These new YANG notifications are encoded and placed within notification messages, which are then queued for egress over the specified transport. The following is an example of an XML encoded notification message over NETCONF transport as per [netconf-notif].


```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <push-update
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>1011</subscription-id>
    <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
    <datastore-contents>
      <foo>
        <bar>some_string</bar>
      </foo>
    </datastore-contents>
  </push-update>
</notification>
```

Figure 1: Push example

The following is an example of an on-change notification. It contains an update for subscription 89, including a new value for a leaf called beta, which is a child of a top-level container called alpha:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <push-change-update xmlns=
    "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>89</subscription-id>
    <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
    <datastore-changes>
      <alpha xmlns="http://example.com/sample-data/1.0" >
        <beta>1500</beta>
      </alpha>
    </datastore-changes>
  </push-change-update>
</notification>
```

Figure 2: Push example for on change

The equivalent update when requesting json encoding:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <push-change-update xmlns=
    "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>89</subscription-id>
    <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
    <datastore-changes>
      {
        "ietf-yang-patch:yang-patch": {
          "patch-id": [
            null
          ],
          "edit": [
            {
              "edit-id": "edit1",
              "operation": "merge",
              "target": "/alpha/beta",
              "value": {
                "beta": 1500
              }
            }
          ]
        }
      }
    </datastore-changes>
  </push-change-update>
</notification>
```

Figure 3: Push example for on change with JSON

When the beta leaf is deleted, the publisher may send

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2015-03-09T19:14:56Z</eventTime>
  <push-change-update xmlns=
    "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>89</subscription-id>
    <time-of-update>2015-03-09T19:14:56.233Z</time-of-update>
    <datastore-changes-xml>
      <alpha xmlns="http://example.com/sample-data/1.0" >
        <beta urn:ietf:params:xml:ns:netconf:base:1.0:
          operation="delete"/>
      </alpha>
    </datastore-changes-xml>
  </push-change-update>
</notification>
```

Figure 4: 2nd push example for on change update

3.8. Subscription management

[subscribe] has been enhanced to support YANG datastore subscription negotiation. These enhancements provide information on why a datastore subscription attempt has failed.

A datastore subscription can be rejected for multiple reasons. This includes the lack of read authorization on a requested data node, or the inability of the publisher push update records as frequently as requested. In such cases, no subscription is established. Instead, the subscription-result with the failure reason is returned as part of the RPC response. As part of this response, a set of alternative subscription parameters MAY be returned that would likely have resulted in acceptance of the subscription request. The subscriber may consider these as part of future subscription attempts.

It should be noted that a rejected subscription does not result in the generation of an rpc-reply with an rpc-error element, as neither the specification of YANG-push specific errors nor the specification of additional data parameters to be returned in an error case are supported as part of a YANG data model.

For instance, for the following request:

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <datastore>push-update</datastore>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <period>500</period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>
```

Figure 5: Establish-Subscription example

the publisher might return:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="http://urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    error-insufficient-resources
  </subscription-result>
  <period>2000</period>
</rpc-reply>
```

Figure 6: Error response example

3.9. Receiver Authorization

A receiver of subscription data MUST only be sent updates for which they have proper authorization. A server MUST ensure that no non-authorized data is included in push updates. To do so, it needs to apply all corresponding checks applicable at the time of a specific pushed update and if necessary silently remove any non-authorized data from subtrees. This enables YANG data pushed based on subscriptions to be authorized equivalently to a regular data retrieval (get) operation.

Alternatively, a server that wants to avoid having to perform filtering of authorized content on each update MAY instead simply reject a subscription request that contains non-authorized data. It MAY subsequently suspend a subscription in case new objects are created during the course of the subscription for which the receiver does not have the necessary authorization, or in case the authorization privileges of a receiver change over the course of the subscription.

The contextual authorization model for data in YANG datastores is the NETCONF Access Control Model [RFC6536bis], Section 3.2.3. However, there are some differences.

One of these clarifications is that datastore selection MUST NOT return continuous errors as part of an on-change subscription. This includes errors such as when there is not read access to every data node specifically named within the filter. Non-authorized data needs to be either simply dropped or, alternatively, the subscription SHOULD be suspended.

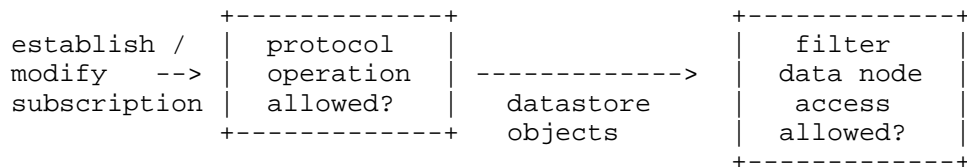


Figure 7: Access control for subscription

Another clarification to [RFC6536bis] is that each of the individual nodes in a resulting update record MUST also have applied access control to resulting pushed messages. This includes validating that read access into new nodes added since the last update record. If read access into previously accessible nodes not explicitly named in the filter are lost mid-subscription, that can be treated as a 'delete' for on-change subscriptions. If not capable of handling such permission changes for dynamic subscriptions, publisher implementations MAY choose to terminate the subscription and to force re-establishment with appropriate filtering.

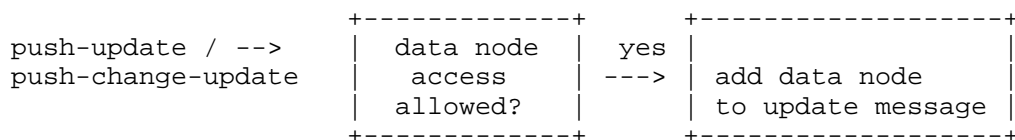


Figure 8: Access control for push updates

3.10. On-change notifiable YANG objects

In some cases, a publisher supporting on-change notifications may not be able to push updates for some object types on-change. Reasons for this might be that the value of the data node changes frequently (e.g., [RFC7223]'s in-octets counter), that small object changes are frequent and meaningless (e.g., a temperature gauge changing 0.1

degrees), or that the implementation is not capable of on-change notification for a particular object.

Support for on-change notification is usually specific to the individual YANG model and/or implementation so it is possible to define in design time. System integrators need this information (without reading any data from a live node).

The default assumption is that no data nodes support on-change notification. Schema nodes and subtrees that support on-change notifications **MUST** be marked by accordingly with the YANG extension "notifiable-on-change". This extension is defined in the data model below.

When an on-change subscription is established, data-nodes are automatically excluded unless they are marked with notifiable-on-change as true. This also means that authorization checks **SHALL NOT** be performed on them. A client can identify which nodes will be included in on-change updated by retrieving the data nodes in the subscription's scope and checking for which notifiable-on-change is marked as true.

Adding notifiable-on-change markings will in general require updating the corresponding YANG models. A simple way to avoid having to modify existing module definitions is to add notifiable-on-change markings by defining module deviations. This means that when a YANG model designer wants to add a notifiable-on-change marking to nodes of an existing module without modifying the module definitions, a new module is introduced that contains deviation statements which add "notifiable-on-change" statements as applicable.

```
deviation /sys:system/sys:system-time {  
  deviate add {  
    yp:notifiable-on-change false;  
  }  
}
```

Figure 9: Deviation Example

3.11. Other considerations

3.11.1. Robustness and reliability

Particularly in the case of on-change push updates, it is important that push updates do not get lost or, in case the loss of an update is unavoidable, that the receiver is notified accordingly.

Update messages for a single subscription **MAY NOT** be resequenced.

It is conceivable that under certain circumstances, a publisher will recognize that it is unable to include within an update record the full set of objects desired per the terms of a subscription. In this case, the publisher **MUST** take one or more of the following actions.

- o A publisher **MUST** set the updates-not-sent flag on any update record which is known to be missing information.
- o It **MAY** choose to suspend a subscription as per [subscribe].
- o When resuming an on-change subscription, the publisher **SHOULD** generate a complete patch from the previous update record. If this is not possible and the synch-on-start option is configured, then the full datastore contents **MAY** be sent instead (effectively replacing the previous contents). If neither of these are possible, then an updates-not-sent flag **MUST** be included on the next push-change-update.

Note: It is perfectly acceptable to have a series of push-change-updates (and even push updates) serially queued at the transport layer awaiting transmission. It is not required to merge pending update messages. I.e., the dampening period applies to update record creation, not transmission.

3.11.2. Update size and fragmentation

Depending on the subscription, the volume of updates can become quite large. Additionally, based on the platform, it is possible that update records for a single subscription are best sent independently from different line-cards. Therefore, it may not always be practical to send the entire update record in a single chunk. Implementations may therefore choose, at their discretion, to "chunk" update records, breaking one subscription's objects across several update records. In this case the updates-not-sent flag will indicate that no single update record is complete, and it is permissible for multiple updates to come into a receiver for a single periodic interval or on-change dampening period.

Care must be taken in chunking as problems may arise for objects that have containment or referential dependencies. The publisher must consider these issues if chunking is provided.

3.11.3. Publisher capacity

It is far preferable to decline a subscription request than to accept such a request when it cannot be met.

Whether or not a subscription can be supported will be determined by a combination of several factors such as the subscription policy (on-change or periodic), the period in which to report changes (1 second periods will consume more resources than 1 hour periods), the amount of data in the subtree that is being subscribed to, and the number and combination of other subscriptions that are concurrently being serviced.

4. A YANG data model for management of datastore push subscriptions

4.1. Overview

The YANG data model for datastore push subscriptions is depicted in the following figure. Following YANG tree convention in the depiction, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. Parentheses with a name in the middle enclose choice and case nodes. New YANG objects defined here (i.e., beyond those from [subscribe]) are identified with "yp".

```

module: ietf-subscribed-notifications
  +--rw filters
  |   +--rw filter* [identifier]
  |   |   +--rw identifier      filter-id
  |   |   +--rw filter-type    filter-type
  |   |   +--rw filter
  |   +--rw subscription-config {configured-subscriptions}?
  |   |   +--rw subscription* [identifier]
  |   |   |   +--rw identifier      subscription-id
  |   |   |   +--rw encoding?      encoding
  |   |   |   +--rw (target)
  |   |   |   |   +--:(event-stream)
  |   |   |   |   |   +--rw stream      stream
  |   |   |   |   +--:(yp:datastore)
  |   |   |   |   |   +--rw yp:datastore  datastore
  |   |   |   +--rw (applied-filter)
  |   |   |   |   +--:(by-reference)
  |   |   |   |   |   +--rw filter-ref    filter-ref
  |   |   |   |   +--:(locally-configured)
  |   |   |   |   |   +--rw filter-type    filter-type
  |   |   |   |   |   +--rw filter
  |   |   |   +--rw stop-time?      yang:date-and-time
  |   |   +--rw receivers
  |   |   |   +--rw receiver* [address port]
  |   |   |   |   +--rw address      inet:host
  |   |   |   |   +--rw port         inet:port-number
  |   |   |   |   +--rw protocol?    transport-protocol
  |   |   +--rw (notification-origin)?

```



```

|      |      +---:(interface-originated)
|      |      |   +---rw source-interface?          if:interface-ref
|      |      +---:(address-originated)
|      |      |   +---rw source-vrf?                string
|      |      |   +---rw source-address             inet:ip-address-no-zone
|      |      +---rw (yp:update-trigger)?
|      |      |   +---:(yp:periodic)
|      |      |   |   +---rw yp:period              yang:timeticks
|      |      |   |   +---rw yp:anchor-time?        yang:date-and-time
|      |      |   +---:(yp:on-change) {on-change}?
|      |      |   |   +---rw yp:dampening-period    yang:timeticks
|      |      |   |   +---rw yp:no-synch-on-start?  empty
|      |      |   |   +---rw yp:excluded-change*    change-type
|      |      +---rw yp:dscp?                        inet:dscp
|      |      +---rw yp:weighting?                   uint8
|      |      +---rw yp:dependency?                  sn:subscription-id
+---ro subscriptions
+---ro subscription* [identifier]
+---ro identifier                      subscription-id
+---ro configured-subscription?
|      |      empty {configured-subscriptions}?
+---ro encoding?                      encoding
+---ro (target)
|      |      +---:(event-stream)
|      |      |   +---ro stream                      stream
|      |      |   +---ro replay-start-time?          yang:date-and-time {replay}?
|      |      +---:(yp:datastore)
|      |      |   +---ro yp:datastore                datastore
+---ro (applied-filter)
|      |      +---:(by-reference)
|      |      |   +---ro filter-ref                  filter-ref
|      |      +---:(locally-configured)
|      |      |   +---ro filter-type                 filter-type
|      |      |   +---ro filter
+---ro stop-time?                      yang:date-and-time
+---ro (notification-origin)?
|      |      +---:(interface-originated)
|      |      |   +---ro source-interface?          if:interface-ref
|      |      +---:(address-originated)
|      |      |   +---ro source-vrf?                string
|      |      |   +---ro source-address             inet:ip-address-no-zone
+---ro receivers
+---ro receiver* [address port]
|      |      +---ro address                        inet:host
|      |      +---ro port                          inet:port-number
|      |      +---ro protocol?                     transport-protocol
|      |      +---ro pushed-notifications?          yang:counter64
|      |      +---ro excluded-notifications?        yang:counter64

```

```

|      +---ro status                                subscription-status
+---ro (yp:update-trigger)?
|   +---:(yp:periodic)
|   |   +---ro yp:period                            yang:timeticks
|   |   +---ro yp:anchor-time?                      yang:date-and-time
|   +---:(yp:on-change) {on-change}?
|   |   +---ro yp:dampening-period                  yang:timeticks
|   |   +---ro yp:no-synch-on-start?                empty
|   |   +---ro yp:excluded-change*                  change-type
+---ro yp:dscp?                                     inet:dscp
+---ro yp:weighting?                               uint8
+---ro yp:dependency?                              sn:subscription-id

rpcs:
+---x establish-subscription
|   +---w input
|   |   +---w encoding?                            encoding
|   |   +---w (target)
|   |   |   +---:(event-stream)
|   |   |   |   +---w stream                        stream
|   |   |   |   +---w replay-start-time?          yang:date-and-time {replay}?
|   |   +---:(yp:datastore)
|   |   |   +---w yp:datastore                      datastore
|   +---w (applied-filter)
|   |   +---:(by-reference)
|   |   |   +---w filter-ref                        filter-ref
|   |   +---:(locally-configured)
|   |   |   +---w filter-type                      filter-type
|   |   |   +---w filter
|   +---w stop-time?                              yang:date-and-time
|   +---w (yp:update-trigger)?
|   |   +---:(yp:periodic)
|   |   |   +---w yp:period                            yang:timeticks
|   |   |   +---w yp:anchor-time?                    yang:date-and-time
|   |   +---:(yp:on-change) {on-change}?
|   |   |   +---w yp:dampening-period                yang:timeticks
|   |   |   +---w yp:no-synch-on-start?              empty
|   |   |   +---w yp:excluded-change*                change-type
|   +---w yp:dscp?                                inet:dscp
|   +---w yp:weighting?                          uint8
|   +---w yp:dependency?                        sn:subscription-id
+---ro output
|   +---ro subscription-result                    subscription-result
|   +---ro (result)?
|   |   +---:(no-success)
|   |   |   +---ro filter-failure?                  string
|   |   |   +---ro replay-start-time-hint?          yang:date-and-time
|   |   |   +---ro yp:period-hint?                  yang:timeticks

```

```

|         |   +--ro yp:error-path?           string
|         |   +--ro yp:object-count-estimate? uint32
|         |   +--ro yp:object-count-limit?   uint32
|         |   +--ro yp:kilobytes-estimate?   uint32
|         |   +--ro yp:kilobytes-limit?      uint32
|         |   +---:(success)
|         |   +--ro identifier                subscription-id
+---x modify-subscription
|   +---w input
|   |   +---w identifier?                    subscription-id
|   |   +---w (applied-filter)
|   |   |   +---:(by-reference)
|   |   |   |   +---w filter-ref            filter-ref
|   |   |   +---:(locally-configured)
|   |   |   |   +---w filter-type           filter-type
|   |   |   |   +---w filter
|   |   +---w stop-time?                    yang:date-and-time
|   |   +---w (yp:update-trigger)?
|   |   |   +---:(yp:periodic)
|   |   |   |   +---w yp:period             yang:timeticks
|   |   |   |   +---w yp:anchor-time?       yang:date-and-time
|   |   |   +---:(yp:on-change) {on-change}?
|   |   |   |   +---w yp:dampening-period   yang:timeticks
|   +---ro output
|   |   +--ro subscription-result           subscription-result
|   |   +--ro (result)?
|   |   |   +---:(no-success)
|   |   |   |   +--ro filter-failure?       string
|   |   |   |   +--ro yp:period-hint?       yang:timeticks
|   |   |   |   +--ro yp:error-path?        string
|   |   |   |   +--ro yp:object-count-estimate? uint32
|   |   |   |   +--ro yp:object-count-limit? uint32
|   |   |   |   +--ro yp:kilobytes-estimate? uint32
|   |   |   |   +--ro yp:kilobytes-limit?   uint32
+---x delete-subscription
|   +---w input
|   |   +---w identifier                    subscription-id
|   +---ro output
|   |   +--ro subscription-result           subscription-result
+---x kill-subscription
|   +---w input
|   |   +---w identifier                    subscription-id
|   +---ro output
|   |   +--ro subscription-result           subscription-result

notifications:
+---n replay-complete
|   +--ro identifier                        subscription-id

```

```

+---n notification-complete
| +--ro identifier      subscription-id
+---n subscription-started
| +--ro identifier      subscription-id
| +--ro encoding?       encoding
| +--ro (target)
| | +--:(event-stream)
| | | +--ro stream      stream
| | | +--ro replay-start-time? yang:date-and-time {replay}?
| | +--:(yp:datastore)
| | | +--ro yp:datastore datastore
| +--ro (applied-filter)
| | +--:(by-reference)
| | | +--ro filter-ref  filter-ref
| | +--:(locally-configured)
| | | +--ro filter-type filter-type
| | | +--ro filter
| +--ro stop-time?      yang:date-and-time
| +--ro (yp:update-trigger)?
| | +--:(yp:periodic)
| | | +--ro yp:period    yang:timeticks
| | | +--ro yp:anchor-time? yang:date-and-time
| | +--:(yp:on-change) {on-change}?
| | | +--ro yp:dampening-period yang:timeticks
| | | +--ro yp:no-synch-on-start? empty
| | | +--ro yp:excluded-change* change-type
| +--ro yp:dscp?        inet:dscp
| +--ro yp:weighting?   uint8
| +--ro yp:dependency?  sn:subscription-id
+---n subscription-resumed
| +--ro identifier      subscription-id
+---n subscription-modified
| +--ro identifier      subscription-id
| +--ro encoding?       encoding
| +--ro (target)
| | +--:(event-stream)
| | | +--ro stream      stream
| | | +--ro replay-start-time? yang:date-and-time {replay}?
| +--ro (applied-filter)
| | +--:(by-reference)
| | | +--ro filter-ref  filter-ref
| | +--:(locally-configured)
| | | +--ro filter-type filter-type
| | | +--ro filter
| +--ro stop-time?      yang:date-and-time
| +--ro (yp:update-trigger)?
| | +--:(yp:periodic)
| | | +--ro yp:period    yang:timeticks

```

```

| | | +--ro yp:anchor-time?          yang:date-and-time
| | | +--:(yp:on-change) {on-change}?
| | |   +--ro yp:dampening-period    yang:timeticks
| | |   +--ro yp:no-synch-on-start?  empty
| | |   +--ro yp:excluded-change*    change-type
| | +--ro yp:dscp?                  inet:dscp
| | +--ro yp:weighting?             uint8
| | +--ro yp:dependency?            sn:subscription-id
+---n subscription-terminated
| +--ro identifier                  subscription-id
| +--ro error-id                    subscription-errors
| +--ro filter-failure?             string
+---n subscription-suspended
| +--ro identifier                  subscription-id
| +--ro error-id                    subscription-errors
| +--ro filter-failure?             string

module: ietf-yang-push
notifications:
+---n push-update
| +--ro subscription-id            sn:subscription-id
| +--ro time-of-update?            yang:date-and-time
| +--ro updates-not-sent?          empty
| +--ro datastore-contents?
+---n push-change-update {on-change}?
| +--ro subscription-id            sn:subscription-id
| +--ro time-of-update?            yang:date-and-time
| +--ro updates-not-sent?          empty
| +--ro datastore-changes?

```

Figure 10: Model structure

Selected components of the model are summarized below.

4.2. Subscription configuration

Both configured and dynamic subscriptions are represented within the list subscription-config. Each subscription has own list elements. New and enhanced parameters extending the basic subscription data model in [subscribe] include:

- o An update filter identifying yang nodes of interest. Filter contents are specified via a reference to an existing filter, or via an in-line definition for only that subscription. This facilitates the reuse of filter definitions, which can be important in case of complex filter conditions. Referenced filters can also allow an implementation to avoid evaluating

filter acceptability during a dynamic subscription request. The case statement differentiates the options.

- o For periodic subscriptions, triggered updates will occur at the boundaries of a specified time interval. These boundaries may be calculated from the periodic parameters:
 - * a "period" which defines duration between period push updates.
 - * an "anchor-time"; update intervals always fall on the points in time that are a multiple of a period after the anchor time. If anchor time is not provided, then the anchor time MUST be set with the creation time of the initial update record.
- o For on-change subscriptions, assuming the dampening period has completed, triggered occurs whenever a change in the subscribed information is detected. On-change subscriptions have more complex semantics that is guided by its own set of parameters:
 - * a "dampening-period" specifies the interval that must pass before a successive update for the subscription is sent. If no dampening period is in effect, the update is sent immediately. If a subsequent change is detected, another update is only sent once the dampening period has passed for this subscription.
 - * an "excluded-change" flag which allows restriction of the types of changes for which updates should be sent (e.g., only add to an update record on object creation).
 - * a "no-synch-on-start" flag which specifies whether a complete update with all the subscribed data is to be sent at the beginning of a subscription.
- o Optional qos parameters to indicate the treatment of a subscription relative to other traffic between publisher and receiver. These include:
 - * A "dscp" QoS marking which MUST be stamped on notification messages to differentiate network QoS behavior.
 - * A "weighting" so that bandwidth proportional to this weighting can be allocated to this subscription relative to others for that receiver.
 - * a "dependency" upon another subscription. Notification messages MUST NOT be sent prior to other notification messages containing update record(s) for the referenced subscription.

- o A subscription's weighting MUST work identically to stream dependency weighting as described within RFC 7540, section 5.3.2.
- o A subscription's dependency MUST work identically to stream dependency as described within RFC 7540, sections 5.3.1, 5.3.3, and 5.3.4. If a dependency is attempted via an RPC, but the referenced subscription does not exist, the dependency will be removed.

4.3. YANG Notifications

4.3.1. Monitoring and OAM Notifications

OAM notifications and mechanism are reused from [subscribe]. Some have been augmented to include the YANG datastore specific objects.

4.3.2. New Notifications for update records

The data model introduces two YANG notifications to encode information for update records: "push-update" and "push-change-update".

"Push-update" is used to send a complete snapshot of the filtered subscription data. This type of notification is used to carry the update records of a periodic subscription. The "push-update" notification is also used with on-change subscriptions for the purposes of allowing a receiver to "synch" on a complete set of subscribed datastore contents. This synching may be done the start of an on-change subscription, and then later in that subscription to force resynchronization. If the "updates-not-sent" flag is set, this indicates that the update record is incomplete.

"Push-change-update" is used to send datastore changes that have occurred in subscribed data since the previous update. This notification is used only in conjunction with on-change subscriptions. This will be encoded as yang-patch data.

If the application detects an informational discontinuity in either notification, the notification MUST include a flag "updates-not-sent". This flag which indicates that not all changes which have occurred since the last update are actually included with this update. In other words, the publisher has failed to fulfill its full subscription obligations. (For example a datastore missed a window in providing objects to a publisher process.) To facilitate synchronization, a publisher MAY subsequently send a push-update containing a full snapshot of subscribed data.

4.4. YANG RPCs

YANG-Push subscriptions are established, modified, and deleted using RPCs augmented from [subscribe].

4.4.1. Establish-subscription RPC

The subscriber sends an establish-subscription RPC with the parameters in section 3.1. An example might look like:

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <period>500</period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>
```

Figure 11: Establish-subscription RPC

The publisher MUST respond explicitly positively (i.e., subscription accepted) or negatively (i.e., subscription rejected) to the request. Positive responses include the subscription-id of the accepted subscription. In that case a publisher MAY respond:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    ok
  </subscription-result>
  <subscription-id
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    52
  </subscription-id>
</rpc-reply>
```

Figure 12: Establish-subscription positive RPC response

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, the lack of read authorization on the requested data node, or the inability of the publisher to provide a stream with the requested semantics.

When the requester is not authorized to read the requested data node, the returned information indicates the node is unavailable. For instance, if the above request was unauthorized to read node "ex:foo" the publisher may return:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    subtree-unavailable
  </subscription-result>
  <filter-failure
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    /ex:foo
  </filter-failure>
</rpc-reply>
```

Figure 13: Establish-subscription access denied response

If a request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error what subscription parameters would have been accepted for the request. However, there are no guarantee that subsequent requests using this info will in fact be accepted.

For example, for the following request:

```
<netconf:rpc message-id="101"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <datastore>running</datastore>
    <filter netconf:type="xpath"
      xmlns:ex="http://example.com/sample-data/1.0"
      select="/ex:foo"/>
    <dampening-period>10</dampening-period>
    <encoding>encode-xml</encoding>
  </establish-subscription>
</netconf:rpc>
```

Figure 14: Establish-subscription request example 2

a publisher that cannot serve on-change updates but periodic updates might return the following:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    period-unsupported
  </subscription-result>
  <period-hint>100</period-hint>
</rpc-reply>
```

Figure 15: Establish-subscription error response example 2

4.4.2. Modify-subscription RPC

The subscriber MAY invoke the modify-subscription RPC for a subscription it previously established. The subscriber will include newly desired values in the modify-subscription RPC. Parameters not included MUST remain unmodified. Below is an example where a subscriber attempts to modify the period of a subscription.

```
<netconf:rpc message-id="102"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <datastore>running</datastore>
    <subscription-id>
      1011
    </subscription-id>
    <period>250</period>
  </modify-subscription>
</netconf:rpc>
```

Figure 16: Modify subscription request

The publisher MUST respond explicitly positively or negatively to the request. A response to a successful modification might look like:

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    ok
  </subscription-result>
</rpc-reply>
```

Figure 17: Modify subscription response

If the subscription modification is rejected, the publisher MUST send a response like it does for an establish-subscription and maintain

the subscription as it was before the modification request. Responses MAY include hints. A subscription MAY be modified multiple times.

A configured subscription cannot be modified using modify-subscription RPC. Instead, the configuration needs to be edited as needed.

4.4.3. Delete-subscription RPC

To stop receiving updates from a subscription and effectively delete a subscription that had previously been established using an establish-subscription RPC, a subscriber can send a delete-subscription RPC, which takes as only input the subscription-id. For example:

```
<netconf:rpc message-id="103"
  xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    <subscription-id>
      1011
    </subscription-id>
  </delete-subscription>
</netconf:rpc>

<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <subscription-result
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
    ok
  </subscription-result>
</rpc-reply>
```

Figure 18: Delete subscription

Configured subscriptions cannot be deleted via RPC, but have to be removed from the configuration.

4.4.4. YANG Module Synchronization

To make subscription requests, the subscriber needs to know the YANG module library available on the publisher. The YANG 1.0 module library information is sent by a NETCONF server in the NETCONF 'hello' message. For YANG 1.1 modules and all modules used with the RESTCONF [RFC8040] protocol, this information is provided by the YANG Library module (ietf-yang-library.yang from [RFC7895]). The YANG

library information is important for the receiver to reproduce the set of object definitions used by the replicated datastore.

The YANG library includes a module list with the name, revision, enabled features, and applied deviations for each YANG module implemented by the publisher. The receiver is expected to know the YANG library information before starting a subscription. The `"/modules-state/module-set-id"` leaf in the `"ietf-yang-library"` module can be used to cache the YANG library information.

The set of modules, revisions, features, and deviations can change at run-time (if supported by the server implementation). In this case, the receiver needs to be informed of module changes before data nodes from changed modules can be processed correctly. The YANG library provides a simple `"yang-library-change"` notification that informs the client that the library has changed. The receiver then needs to re-read the entire YANG library data for the replicated server in order to detect the specific YANG library changes. The `"ietf-netconf-notifications"` module defined in [RFC6470] contains a `"netconf-capability-change"` notification that can identify specific module changes. For example, the module URI capability of a newly loaded module will be listed in the `"added-capability"` leaf-list, and the module URI capability of an removed module will be listed in the `"deleted-capability"` leaf-list.

5. YANG module

```
<CODE BEGINS>; file "ietf-yang-push@2017-06-26.yang"
module ietf-yang-push {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-yang-types {
    prefix yang;
  }
  import ietf-subscribed-notifications {
    prefix sn;
  }

  organization "IETF";
  contact
    "WG Web:  <http://tools.ietf.org/wg/netconf/>
    WG List:  <mailto:netconf@ietf.org>
```

WG Chair: Mahesh Jethanandani
<mailto:mjethanandani@gmail.com>

WG Chair: Mehmet Ersue
<mailto:mehmet.ersue@nokia.com>

Editor: Alexander Clemm
<mailto:ludwig@clemm.org>

Editor: Eric Voit
<mailto:evoit@cisco.com>

Editor: Alberto Gonzalez Prieto
<mailto:albertgo@cisco.com>

Editor: Ambika Prasad Tripathy
<mailto:ambtripa@cisco.com>

Editor: Einar Nilsen-Nygaard
<mailto:einarnn@cisco.com>

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

Editor: Balazs Lengyel
<mailto:balazs.lengyel@ericsson.com>";

description

"This module contains conceptual YANG specifications
for YANG push.";

revision 2017-06-26 {

description

"Move to identities for filters, datastores.";

reference

"YANG Datastore Push, draft-ietf-netconf-yang-push-07";

}

/*

* EXTENSIONS

*/

extension notifiable-on-change {

argument "value";

description

"Indicates whether changes to the data node are reportable in
on-change subscriptions.

The statement MUST only be a substatement of the leaf, leaf-list, container, list, anyxml, anydata statements. Zero or One notifiable-on-change statement is allowed per parent statement. NO substatements are allowed.

The argument is a boolean value indicating whether on-change notifications are supported. If notifiable-on-change is not specified, the default is the same as the parent data node's value. For top level data nodes the default value is false.";

```
}
/*
 * FEATURES
 */

feature on-change {
  description
    "This feature indicates that on-change updates are
    supported.";
}

/*
 * IDENTITIES
 */

/* Error type identities for datastore subscription */
identity period-unsupported {
  base sn:error;
  description
    "Requested time period is too short. This can be for both
    periodic and on-change dampening.";
}

identity qos-unsupported {
  base sn:error;
  description
    "Subscription QoS parameters not supported on this platform.";
}

identity dscp-unavailable {
  base sn:error;
  description
    "Requested DSCP marking not allocatable.";
}

identity on-change-unsupported {
  base sn:error;
  description
```

```
    "On-change not supported.";
}

identity synch-on-start-unsupported {
    base sn:error;
    description
        "On-change synch-on-start not supported.";
}

identity synch-on-start-datatree-size {
    base sn:error;
    description
        "Synch-on-start would push a datatree which exceeds size limit.";
}

identity reference-mismatch {
    base sn:error;
    description
        "Mismatch in filter key and referenced yang subtree.";
}

identity data-unavailable {
    base sn:error;
    description
        "Referenced yang node or subtree doesn't exist, or read
        access is not permitted.";
}

identity datatree-size {
    base sn:error;
    description
        "Resulting push updates would exceed size limit.";
}

/* Datastore identities */
identity datastore {
    description
        "A datastore.";
}
identity candidate {
    base datastore;
    description
        "The candidate datastore per RFC-6241.";
    reference "RFC-6241, #5.1";
}
identity running {
    base datastore;
    description
```

```
    "The running datastore per RFC-6241.";
    reference "RFC-6241, #5.1";
}
identity startup {
    base datastore;
    description
        "The startup datastore per RFC-6241.";
    reference "RFC-6241, #5.1";
}
identity operational {
    base datastore;
    description
        "The operational datastore contains all configuration data
        actually used by the system, including all applied configuration,
        system-provided configuration and values defined by any supported
        data models. In addition, the operational datastore also
        contains state data.";
    reference
        "the original text came from draft-ietf-netmod-revised-datastores
        -01, section #4.3. This definition is expected to remain stable
        meaning later reconciliation between the drafts unnecessary.";
}

/* New filter identities (adds to 'sn') */
identity subtree {
    base sn:filter;
    description
        "A filter which follows the subtree filter syntax specified
        in RFC 6241.";
    reference "RFC 6241 section 6";
}

/*
 * TYPE DEFINITIONS
 */

typedef change-type {
    type enumeration {
        enum "create" {
            description
                "Create a new data resource if it does not already exist. If
                it already exists, replace.";
        }
        enum "delete" {
            description
                "Delete a data resource if it already exists. If it does not
                exists, take no action.";
        }
    }
}
```



```

    enum "insert" {
        description
            "Insert a new user-ordered data resource";
    }
    enum "merge" {
        description
            "merge the edit value with the target data resource; create
            if it does not already exist";
    }
    enum "move" {
        description
            "Reorder the target data resource";
    }
    enum "replace" {
        description
            "Replace the target data resource with the edit value";
    }
    enum "remove" {
        description
            "Remove a data resource if it already exists ";
    }
}
description
    "Specifies different types of datastore changes.";
reference
    "RFC 8072 section 2.5, with a delta that it is ok to receive
    ability create on an existing node, or recieve a delete on a
    missing node.";
}

typedef datastore {
    type identityref {
        base datastore;
    }
    description
        "Specifies a system-provided datastore. May also specify ability
        portion of a datastore, so as to reduce the filtering effort.";
}

/*
 * GROUP DEFINITIONS
 */

grouping datastore-criteria {
    description
        "A reusable place to define the meaning of datastore.";
    leaf datastore {
        type datastore;
    }
}

```

```
    mandatory true;
    description
        "Datastore against which the subscription has been applied.";
}
}
```

```
grouping update-policy-modifiable {
    description
        "This grouping describes the datastore specific subscription
        conditions that can be changed during the lifetime of the
        subscription.";
    choice update-trigger {
        description
            "Defines necessary conditions for sending an event to
            the subscriber.";
        case periodic {
            description
                "The agent is requested to notify periodically the current
                values of the datastore as defined by the filter.";
            leaf period {
                type yang:timeticks;
                mandatory true;
                description
                    "Duration of time which should occur between periodic
                    push updates. Where the anchor of a start-time is
                    available, the push will include the objects and their
                    values which exist at an exact multiple of timeticks
                    aligning to this start-time anchor.";
            }
            leaf anchor-time {
                type yang:date-and-time;
                description
                    "Designates a timestamp from which the series of periodic
                    push updates are computed. The next update will take place
                    at the next period interval from the anchor time. For
                    example, for an anchor time at the top of a minute and a
                    period interval of a minute, the next update will be sent
                    at the top of the next minute.";
            }
        }
    }
    case on-change {
        if-feature "on-change";
        description
            "The agent is requested to notify changes in values in the
            datastore subset as defined by a filter.";
        leaf dampening-period {
            type yang:timeticks;
            mandatory true;
        }
    }
}
```

```

        description
            "The shortest time duration which is allowed between the
              creation of independent yang object update messages.
              Effectively this is the amount of time that needs to hav
e              passed since the last update.";
    }
}
}

grouping update-policy {
    description
        "This grouping describes the datastore specific subscription
        conditions of a subscription.";
    uses update-policy-modifiable {
        augment "update-trigger/on-change" {
            description
                "Includes objects not modifiable once subscription is
                established.";
            leaf no-synch-on-start {
                type empty;
                description
                    "This leaf acts as a flag that determines behavior at the
                    start of the subscription. When present, synchronization
                    of state at the beginning of the subscription is outside
                    the scope of the subscription. Only updates about changes
                    that are observed from the start time, i.e. only push-
                    change-update notifications are sent. When absent (default
                    behavior), in order to facilitate a receiver's
                    synchronization, a full update is sent when the
                    subscription starts using a push-update notification, just
                    like in the case of a periodic subscription. After that,
                    push-change-update notifications only are sent unless the
                    Publisher chooses to resynch the subscription again.";
            }
            leaf-list excluded-change {
                type change-type;
                description
                    "Use to restrict which changes trigger an update.
                    For example, if modify is excluded, only creation and
                    deletion of objects is reported.";
            }
        }
    }
}

grouping update-qos {
    description

```

```
    "This grouping describes Quality of Service information
    concerning a subscription. This information is passed to lower
    layers for transport prioritization and treatment";
  leaf dscp {
    type inet:dscp;
    default "0";
    description
      "The push update's IP packet transport priority. This is made
      visible across network hops to receiver. The transport
      priority is shared for all receivers of a given subscription.";
  }
  leaf weighting {
    type uint8 {
      range "0 .. 255";
    }
    description
      "Relative weighting for a subscription. Allows an underlying
      transport layer perform informed load balance allocations
      between various subscriptions";
    reference
      "RFC-7540, section 5.3.2";
  }
  leaf dependency {
    type sn:subscription-id;
    description
      "Provides the Subscription ID of a parent subscription which
      has absolute priority should that parent have push updates
      ready to egress the publisher. In other words, there should be
      no streaming of objects from the current subscription if of
      the parent has something ready to push.";
    reference
      "RFC-7540, section 5.3.1";
  }
}

grouping update-error-hints {
  description
    "Allow return additional negotiation hints that apply
    specifically to push updates.";
  leaf period-hint {
    type yang:timeticks;
    description
      "Returned when the requested time period is too short. This
      hint can assert an viable period for both periodic push
      cadence and on-change dampening.";
  }
  leaf error-path {
    type string;
  }
}
```

```
    description
      "Reference to a YANG path which is associated with the error
       being returned.";
  }
  leaf object-count-estimate {
    type uint32;
    description
      "If there are too many objects which could potentially be
       returned by the filter, this identifies the estimate of the
       number of objects which the filter would potentially pass.";
  }
  leaf object-count-limit {
    type uint32;
    description
      "If there are too many objects which could be returned by the
       filter, this identifies the upper limit of the publisher's
       ability to service for this subscription.";
  }
  leaf kilobytes-estimate {
    type uint32;
    description
      "If the returned information could be beyond the capacity of
       the publisher, this would identify the data size which could
       result from this filter.";
  }
  leaf kilobytes-limit {
    type uint32;
    description
      "If the returned information would be beyond the capacity of
       the publisher, this identifies the upper limit of the
       publisher's ability to service for this subscription.";
  }
}

/*
 * DATA NODES
 */

augment "/sn:establish-subscription/sn:input" {
  description
    "This augmentation adds additional subscription parameters that
     apply specifically to datastore updates to RPC input.";
  uses update-policy;
  uses update-qos;
}
augment "/sn:establish-subscription/sn:input/sn:target" {
  description
    "This augmentation adds the datastore as a valid parameter object
```

```
    for the subscription to RPC input.  This provides a target for
    the filter.";
  case datastore {
    uses datastore-criteria;
  }
}
augment "/sn:establish-subscription/sn:output/"+
  "sn:result/sn:no-success" {
  description
    "This augmentation adds datastore specific error info
    and hints to RPC output.";
  uses update-error-hints;
}
augment "/sn:modify-subscription/sn:input" {
  description
    "This augmentation adds additional subscription parameters
    specific to datastore updates.";
  uses update-policy-modifiable;
}
augment "/sn:modify-subscription/sn:output/"+
  "sn:result/sn:no-success" {
  description
    "This augmentation adds push datastore error info and hints to
    RPC output.";
  uses update-error-hints;
}

notification push-update {
  description
    "This notification contains a push update, containing data
    subscribed to via a subscription. This notification is sent for
    periodic updates, for a periodic subscription. It can also be
    used for synchronization updates of an on-change subscription.
    This notification shall only be sent to receivers of a
    subscription; it does not constitute a general-purpose
    notification.";
  leaf subscription-id {
    type sn:subscription-id;
    mandatory true;
    description
      "This references the subscription because of which the
      notification is sent.";
  }
  leaf time-of-update {
    type yang:date-and-time;
    description
      "This leaf contains the time of the update.";
  }
}
```

```
leaf updates-not-sent {
  type empty;
  description
    "This is a flag which indicates that not all data nodes
    subscribed to are included with this update. In other words,
    the publisher has failed to fulfill its full subscription
    obligations. This may lead to intermittent loss of
    synchronization of data at the client. Synchronization at the
    client can occur when the next push-update is received.";
}
anydata datastore-contents {
  description
    "This contains the updated data. It constitutes a snapshot
    at the time-of-update of the set of data that has been
    subscribed to. The format and syntax of the data
    corresponds to the format and syntax of data that would be
    returned in a corresponding get operation with the same
    filter parameters applied.";
}
}
notification push-change-update {
  if-feature "on-change";
  description
    "This notification contains an on-change push update. This
    notification shall only be sent to the receivers of a
    subscription; it does not constitute a general-purpose
    notification.";
  leaf subscription-id {
    type sn:subscription-id;
    mandatory true;
    description
      "This references the subscription because of which the
      notification is sent.";
  }
  leaf time-of-update {
    type yang:date-and-time;
    description
      "This leaf contains the time of the update, i.e. the time at
      which the change was observed.";
  }
  leaf updates-not-sent {
    type empty;
    description
      "This is a flag which indicates that not all changes which
      have occurred since the last update are included with this
      update. In other words, the publisher has failed to
      fulfill its full subscription obligations, for example in
      cases where it was not able to keep up with a change burst.
```

```

    To facilitate synchronization, a publisher may subsequently
    send a push-update containing a full snapshot of subscribed
    data. Such a push-update might also be triggered by a
    subscriber requesting an on-demand synchronization.";
  }
  anydata datastore-changes {
    description
      "This contains datastore contents that has changed since the
      previous update, per the terms of the subscription. Changes
      are encoded analogous to the syntax of a corresponding yang-
      patch operation, i.e. a yang-patch operation applied to the
      YANG datastore implied by the previous update to result in the
      current state (and assuming yang-patch could also be applied
      to operational data).";
  }
}

augment "/sn:subscription-started" {
  description
    "This augmentation adds many yang datastore specific objects to
    the notification that a subscription has started.";
  uses update-policy;
  uses update-qos;
}
augment "/sn:subscription-started/sn:target" {
  description
    "This augmentation allows the datastore to be included as part
    of the notification that a subscription has started.";
  case datastore {
    uses datastore-criteria;
  }
}
augment "/sn:subscription-modified" {
  description
    "This augmentation adds many yang datastore specific objects to
    the notification that a subscription has been modified.";
  uses update-policy;
  uses update-qos;
}

augment "/sn:subscription-config/sn:subscription" {
  description
    "This augmentation adds many yang datastore specific objects
    which can be configured as opposed to established via RPC.";
  uses update-policy;
  uses update-qos;
}
augment "/sn:subscription-config/sn:subscription/sn:target" {
```



```
    description
      "This augmentation adds the datastore to the filtering
      criteria for a subscription.";
    case datastore {
      uses datastore-criteria;
    }
  }
  augment "/sn:subscriptions/sn:subscription" {
    yp:notifiable-on-change true;
    description
      "This augmentation adds many datastore specific objects to a
      subscription.";
    uses update-policy;
    uses update-qos;
  }
  augment "/sn:subscriptions/sn:subscription/sn:target" {
    description
      "This augmentation allows the datastore to be displayed as part
      of the filtering criteria for a subscription.";
    case datastore {
      uses datastore-criteria;
    }
  }
}
/* YANG Parser Pyang crashing below, due to fixed bug
   https://github.com/mbj4668/pyang/issues/300

deviation "/sn:subscriptions/sn:subscription/sn:receivers/"
  + "sn:receiver/sn:pushed-notifications" {
  deviate add {
    yp:notifiable-on-change false;
  }
}
deviation "/sn:subscriptions/sn:subscription/sn:receivers/"
  + "sn:receiver/sn:excluded-notifications" {
  deviate add {
    yp:notifiable-on-change false;
  }
}
YANG Parser Pyang crashing on the following syntax above */
}
<CODE ENDS>
```

6. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-yang-push
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

This document registers the following YANG module in the "YANG Module Names" registry [RFC6020]:

Name: ietf-yang-push
Namespace: urn:ietf:params:xml:ns:yang:ietf-yang-push
Prefix: yp
Reference: draft-ietf-netconf-yang-push-07.txt (RFC form)

7. Security Considerations

All security considerations from [subscribe] are relevant for datastores. In addition there are specific security considerations for receivers defined in Section 3.9

If the access control permissions on subscribed YANG nodes change during the lifecycle of a subscription, a publisher MUST either transparently conform to the new access control permissions, or must terminate or restart the subscriptions so that new access control permissions are re-established.

The NETCONF Authorization Control Model SHOULD be used to restrict the delivery of YANG nodes for which the receiver has no access.

8. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Tim Jenkins, Kent Watsen, Susan Hares, Yang Geng, Peipei Guo, Michael Scharf, Sharon Chisholm, and Guangying Zheng.

9. References

9.1. Normative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [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>>.

- [RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <<http://www.rfc-editor.org/info/rfc6470>>.
- [RFC6536bis] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", draft-ietf-netconf-rfc6536bis-01 (work in progress), March 2017.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, <<http://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, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<http://www.rfc-editor.org/info/rfc7951>>.
- [RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <<http://www.rfc-editor.org/info/rfc8072>>.
- [subscribe] Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Notifications", draft-ietf-netconf-subscribed-notifications-01 (work in progress), April 2017.

9.2. Informative References

- [http-notif] Voit, E., Gonzalez Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Clemm, A., and A. Bierman, "Restconf and HTTP Transport for Event Notifications", March 2017.
- [netconf-notif] Gonzalez Prieto, A., Clemm, A., Voit, E., Tripathy, A., Nilsen-Nygaard, E., Chisholm, S., and H. Trevino, "NETCONF Support for Event Notifications", October 2016.
- [notifications2] Voit, E., Bierman, A., Clemm, A., and T. Jenkins, "YANG Notification Headers and Bundles", April 2017.

- [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>>.
- [RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, DOI 10.17487/RFC7223, May 2014, <<http://www.rfc-editor.org/info/rfc7223>>.
- [RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <<http://www.rfc-editor.org/info/rfc7923>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

Appendix A. Relationships to other drafts

There are other related drafts which are progressing in the NETCONF WG. This section details the relationship of this draft to those others.

A.1. ietf-netconf-subscribed-notifications

The draft [subscribe] is the technical foundation around which the rest of the YANG push datastore specific mechanisms are layered.

A.2. ietf-netconf-netconf-event-notif

The [netconf-notif] draft supports yang-push by defining NETCONF transport specifics. Included are:

- o bindings for RPC communications and Event Notifications over NETCONF.
- o encoded examples

A.3. ietf-netconf-restconf-notif

The [http-notif] draft supports yang-push by defining transport specific guidance where some form of HTTP is used underneath. Included are:

- o bindings for RPC communications over RESTCONF
- o bindings for Event Notifications over HTTP2 and HTTP1.1

- o encoded examples
- o end-to-end deployment guidance for Call Home and TLS Heartbeat

A.4. voit-notifications2

The draft [notifications2] is not required to implement yang-push. Instead it defines data plane notification elements which improve the delivered experience. The following capabilities are specified:

- o Defines common encapsulation headers objects to support functionality such as event severity, message signing, message loss discovery, message de-duplication, originating process identification.
- o Defines how to bundle multiple event records into a single notification message.

These capabilities would be delivered by adding the drafts newly proposed header objects to the push-update and push-change-update notifications defined here. This draft is not yet adopted by the NETCONF WG.

Appendix B. Technologies to be considered for future iterations

B.1. Proxy YANG Subscription when the Subscriber and Receiver are different

The properties of Dynamic and Configured Subscriptions can be combined to enable deployment models where the Subscriber and Receiver are different. Such separation can be useful with some combination of:

- o An operator does not want the subscription to be dependent on the maintenance of transport level keep-alives. (Transport independence provides different scalability characteristics.)
- o There is not a transport session binding, and a transient Subscription needs to survive in an environment where there is unreliable connectivity with the Receiver and/or Subscriber.
- o An operator wants the Publisher to include highly restrictive capacity management and Subscription security mechanisms outside of domain of existing operational or programmatic interfaces.

To build a Proxy Subscription, first the necessary information must be signaled as part of the <establish-subscription>. Using this set

of Subscriber provided information; the same process described within section 3 will be followed.

After a successful establishment, if the Subscriber wishes to track the state of Receiver subscriptions, it may choose to place a separate on-change Subscription into the "Subscriptions" subtree of the YANG Datastore on the Publisher.

B.2. OpState and Filters

Currently there are ongoing discussions to revise the concept of datastores, allowing for proper handling and distinction of intended versus applied configurations and extending the notion of a datastore to operational data. When finalized, the new concept may open up the possibility for new types of subscription filters, for example, targeting specific datastores and targeting (potentially) differences in datatrees across different datastores.

Likewise, it is conceivable that filters are defined that apply to metadata, such as data nodes for which metadata has been defined that meets a certain criteria.

Defining any such subscription filters at this point would be highly speculative in nature. However, it should be noted that corresponding extensions may be defined in future specifications. Any such extensions will be straightforward to accommodate by introducing a model that defines new filter types, and augmenting the new filter type into the subscription model.

B.3. Splitting push updates

Push updates may become fairly large and extend across multiple subsystems in a YANG-Push Server. As a result, it is conceivable to not combine all updates into a single update message, but to split updates into multiple separate update messages. Such splitting could occur along multiple criteria: limiting the number of data nodes contained in a single update, grouping updates by subtree, grouping updates by internal subsystems (e.g., by line card), or grouping them by other criteria.

Splitting updates bears some resemblance to fragmenting packets. In effect, it can be seen as fragmenting update messages at an application level. However, from a transport perspective, splitting of update messages is not required as long as the transport does not impose a size limitation or provides its own fragmentation mechanism if needed. We assume this to be the case for YANG-Push. In the case of NETCONF, RESTCONF, HTTP/2, no limit on message size is imposed.

In case of other transports, any message size limitations need to be handled by the corresponding transport mapping.

There may be some scenarios in which splitting updates might still make sense. For example, if updates are collected from multiple independent subsystems, those updates could be sent separately without need for combining. However, if updates were to be split, other issues arise. Examples include indicating the number of updates to the receiver, distinguishing a missed fragment from a missed update, and the ordering with which updates are received. Proper addressing those issues would result in considerable complexity, while resulting in only very limited gains. In addition, if a subscription is found to result in updates that are too large, a publisher can always reject the request for a subscription while the subscriber is always free to break a subscription up into multiple subscriptions.

B.4. Potential Subscription Parameters

A possible is the introduction of an additional parameter "changes-only" for periodic subscription. Including this flag would result in sending at the end of each period an update containing only changes since the last update (i.e. a change-update as in the case of an on-change subscription), not a full snapshot of the subscribed information. Such an option might be interesting in case of data that is largely static and bandwidth-constrained environments.

Appendix C. Issues that are currently being worked and resolved

(To be removed by RFC editor prior to publication)

Issue #6: Data plane notifications and layered headers. Specifically how do we want to enable standard header unification and bundle support vs. the data plane notifications currently defined.

Appendix D. Changes between revisions

(To be removed by RFC editor prior to publication)

v06 - v07

- o Clarifying text tweaks.
- o Clarification that filters act as selectors for subscribed data nodes; support for value filters not included but possible as a future extension
- o Filters don't have to be matched to existing YANG objects

v05 - v06

- o Security considerations updated.
- o Base YANG model in [sn] updated as part of move to identities, YANG augmentations in this doc matched up
- o Terms refined and text updates throughout
- o Appendix talking about relationship to other drafts added.
- o Datastore replaces stream
- o Definitions of filters improved

v04 to v05

- o Referenced based subscription document changed to Subscribed Notifications from 5277bis.
- o Getting operational data from filters
- o Extension notifiable-on-change added
- o New appendix on potential futures. Moved text into there from several drafts.
- o Subscription configuration section now just includes changed parameters from Subscribed Notifications
- o Subscription monitoring moved into Subscribed Notifications
- o New error and hint mechanisms included in text and in the yang model.
- o Updated examples based on the error definitions
- o Groupings updated for consistency
- o Text updates throughout

v03 to v04

- o Updates-not-sent flag added
- o Not notifiable extension added
- o Dampening period is for whole subscription, not single objects

- o Moved start/stop into rfc5277bis
- o Client and Server changed to subscriber, publisher, and receiver
- o Anchor time for periodic
- o Message format for synchronization (i.e. synch-on-start)
- o Material moved into 5277bis
- o QoS parameters supported, by not allowed to be modified by RPC
- o Text updates throughout

Authors' Addresses

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Alberto Gonzalez Prieto

Email: albert.gonzalezprieto@yahoo.com

Ambika Prasad Tripathy
Cisco Systems

Email: ambtripa@cisco.com

Einar Nilsen-Nygaard
Cisco Systems

Email: einarnn@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Balazs Lengyel
Ericsson

Email: balazs.lengyel@ericsson.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 21, 2017

K. Watsen
Juniper Networks
M. Abrahamsson
T-Systems
I. Farrer
Deutsche Telekom AG
June 19, 2017

Zero Touch Provisioning for NETCONF or RESTCONF based Management
draft-ietf-netconf-zerotouch-14

Abstract

This draft presents a secure technique for establishing a NETCONF or RESTCONF connection between a newly deployed device, configured with just its factory default settings, and its deployment specific network management system (NMS).

Editorial Note (To be removed by RFC Editor)

This draft contains many placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. Please note that no other RFC Editor instructions are specified anywhere else in this document.

Artwork in the IANA Considerations section contains placeholder values for DHCP options pending IANA assignment. Please apply the following replacements:

- o "OPTION_V4_ZEROTOUCH_REDIRECT" --> the option code assigned for the "DHCPv4 Zero Touch Option" option
- o "OPTION_V6_ZEROTOUCH_REDIRECT" --> the option code assigned for the "DHCPv6 Zero Touch Option" option

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- o "XXXX" --> the assigned RFC value for this draft

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- o "2017-06-19" --> the publication date of this draft

Please update the following references to reflect their final RFC assignments:

- o I-D.ieft-netconf-netconf-client-server
- o I-D.ietf-anima-bootstrapping-keyinfra

The following one Appendix section is to be removed prior to publication:

- o Appendix A. Change Log

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 December 21, 2017.

Copyright Notice

Copyright (c) 2017 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	4
1.1. Use Cases	5
1.2. Terminology	5
1.3. Requirements Language	7
1.4. Tree Diagram Notation	7
2. Types of Bootstrapping Information	8
2.1. Redirect Information	8
2.2. Onboarding Information	9
3. Artifacts	9
3.1. Zero Touch Information	10
3.2. Owner Certificate	10
3.3. Ownership Voucher	11
4. Artifact Groupings	11
4.1. Unsigned Information	12
4.2. Signed Information (without Revocations)	12
4.3. Signed Information (with Revocations)	13
5. Sources of Bootstrapping Data	13
5.1. Removable Storage	13
5.2. DNS Server	14
5.3. DHCP Server	15
5.4. Bootstrap Server	16
6. Workflow Overview	18
6.1. Enrollment and Ordering Devices	18
6.2. Owner Stages the Network for Bootstrap	20
6.3. Device Powers On	22
7. Device Details	25
7.1. Factory Default State	25
7.2. Boot Sequence	26
7.3. Processing a Source of Bootstrapping Data	27
7.4. Validating Signed Data	29
7.5. Processing Redirect Information	30
7.6. Processing Onboarding Information	30
8. The Zero Touch Information Artifact	31
8.1. Tree Diagram	31
8.2. Example Usage	32
8.3. YANG Module	35
9. The Zero Touch Bootstrap Server API	40
9.1. Tree Diagram	40
9.2. Example Usage	41
9.3. YANG Module	44
10. DHCP Zero Touch Options	52
10.1. DHCPv4 Zero Touch Option	52
10.2. DHCPv6 Zero Touch Option	53
10.3. Common Field Encoding	54
11. Security Considerations	55
11.1. Immutable storage for trust anchors	55

11.2.	Clock Sensitivity	55
11.3.	Blindly authenticating a bootstrap server	56
11.4.	Entropy loss over time	56
11.5.	Serial Numbers	56
11.6.	Sequencing Sources of Bootstrapping Data	56
12.	IANA Considerations	56
12.1.	The BOOTP Manufacturer Extensions and DHCP Options Registry	56
12.2.	The IETF XML Registry	57
12.3.	The YANG Module Names Registry	57
13.	Other Considerations	57
14.	Acknowledgements	58
15.	References	58
15.1.	Normative References	58
15.2.	Informative References	60
Appendix A.	Change Log	62
A.1.	ID to 00	62
A.2.	00 to 01	62
A.3.	01 to 02	62
A.4.	02 to 03	63
A.5.	03 to 04	63
A.6.	04 to 05	63
A.7.	05 to 06	64
A.8.	06 to 07	64
A.9.	07 to 08	64
A.10.	08 to 09	64
A.11.	09 to 10	64
A.12.	10 to 11	65
A.13.	11 to 12	65
A.14.	12 to 13	65
A.15.	13 to 14	66
Authors' Addresses	66

1. Introduction

A fundamental business requirement for any network operator is to reduce costs where possible. For network operators, deploying devices to many locations can be a significant cost, as sending trained specialists to each site for installations is both cost prohibitive and does not scale.

This document defines a bootstrapping strategy enabling devices to securely obtain bootstrapping data with no installer action beyond physical placement and connecting network and power cables. The ultimate goal of this document is to enable a secure NETCONF [RFC6241] or RESTCONF [RFC8040] connection to a deployment specific network management system (NMS).

1.1. Use Cases

- o Device connecting to a remotely administered network

This use-case involves scenarios, such as a remote branch office or convenience store, whereby a device connects as an access gateway to an ISP's network. Assuming it is not possible to customize the ISP's network to provide any bootstrapping support, and with no other nearby device to leverage, the device has no recourse but to reach out to an Internet-based bootstrap server to bootstrap off of.

- o Device connecting to a locally administered network

This use-case covers all other scenarios and differs only in that the device may additionally leverage nearby devices, which may direct it to use a local service to bootstrap off of. If no such information is available, or the device is unable to use the information provided, it can then reach out to network just as it would for the remotely administered network use-case.

1.2. Terminology

This document uses the following terms (sorted by name):

Artifact: The term "artifact" is used throughout to represent any of the three artifacts defined in Section 3 (Zero Touch Information, Ownership Voucher, and Owner Certificate). These artifacts collectively provide all the bootstrapping data a device may use.

Bootstrapping Data: The term "bootstrapping data" is used throughout this document to refer to the collection of data that a device may obtain from any source of bootstrapping data. Specifically, it refers to the artifacts defined in Section 3.

Bootstrap Server: The term "bootstrap server" is used within this document to mean any RESTCONF server implementing the YANG module defined in Section 9.3.

Device: The term "device" is used throughout this document to refer to the network element that needs to be bootstrapped. See Section 7 for more information about devices.

Initial Secure Device Identifier (IDevID): The term "IDevID" is defined in [Std-802.1AR-2009] as the secure device identifier (DevID) installed on the device by the manufacturer. This

identifier is used in this document to enable a Bootstrap Server to securely identify and authenticate a device.

Manufacturer: The term "manufacturer" is used herein to refer to the manufacturer of a device or a delegate of the manufacturer.

Network Management System (NMS): The acronym "NMS" is used throughout this document to refer to the deployment specific management system that the bootstrapping process is responsible for introducing devices to. From a device's perspective, when the bootstrapping process has completed, the NMS is a NETCONF or RESTCONF client.

Onboarding Information: The term "onboarding information" is used herein to refer to one of the two types of 'zero touch information' defined in this document, the other being 'redirect information'. Specifically, onboarding information guides a device to, for instance, install a specific boot-image and commit a specific configuration.

Owner: The term "owner" is used throughout this document to refer to the person or organization that purchased or otherwise owns a device.

Owner Certificate: The term "owner certificate" is used in this document to represent an X.509 certificate that binds an owner identity to a public key, which a device can use to validate a signature over the zero touch information artifacts. The owner certificate is one of the three bootstrapping artifacts described in Section 3.

Ownership Voucher: The term "ownership voucher" is used in this document to represent the voucher artifact defined in [I-D.ietf-anima-voucher]. The ownership voucher is used to assign a device to an owner. The ownership voucher is one of the three bootstrapping artifacts described in Section 3.

Redirect Information: The term "redirect information" is used herein to refer to one of the two types of 'zero touch information' defined in this document, the other being 'onboarding information'. Specifically, redirect information directs a device to connect to a specified bootstrap server.

Redirect Server: The term "redirect server" is used to refer to a bootstrap server that only returns redirect information. A redirect server is particularly useful when hosted by a manufacturer, as an Internet-based resource to redirect devices to deployment-specific bootstrap servers.

Signed Data: The term "signed data" is used throughout to mean either redirect information or onboarding information that has been signed, specifically by a private key possessed by a device's owner.

Unsigned Data: The term "unsigned data" is used throughout to mean either redirect information or onboarding information that has not been signed.

Zero Touch Information: The term "zero touch information" is used generally herein to refer either redirect information or onboarding information. Zero touch information is one of the three bootstrapping artifacts described in Section 3.

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.4. Tree Diagram Notation

A simplified graphical representation of the data models is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Braces "{" and "}" enclose feature names, and indicate that the named feature must be present for the subtree to be present.
- o Abbreviations before data node names: "rw" (read-write) represents configuration data and "ro" (read-only) represents state data.
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Types of Bootstrapping Information

This document defines two types of information that devices access during the bootstrapping process. These information types are described in this section. Examples are provided in Section 8.2

2.1. Redirect Information

Redirect information redirects a device to another bootstrap server. Redirect information encodes a list of bootstrap servers, each defined by its hostname or IP address, an optional port, and an optional trust anchor certificate.

Redirect information is YANG modeled data formally defined by the "redirect-information" container in the YANG module presented in Section 8.3. This container has the tree diagram shown below. Please see Section 1.4 for tree diagram notation.

```

+---:(redirect-information)
  +--ro redirect-information
    +--ro bootstrap-server* [address]
      +--ro address          inet:host
      +--ro port?            inet:port-number
      +--ro trust-anchor?    binary
```

Redirect information MAY be trusted or untrusted. The redirect information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device's owner. In all other cases, the redirect information is untrusted.

Trusted redirect information is useful for enabling a device to establish a secure connection to a bootstrap server, which is possible when the redirect information includes the bootstrap server's trust anchor certificate. When a device is able to establish a secure connection to a bootstrap server, the data is implicitly trusted, and does not need to be signed.

Untrusted redirect information is useful for directing a device to a bootstrap server where signed data has been staged for it to obtain. When the redirect information is untrusted, the device MUST discard any potentially included trust anchor certificates and SHOULD establish a provisional connection (by blindly accepting the TLS certificate) to any of the specified bootstrap servers. In this case, the device MUST NOT trust the bootstrap server, and data provided by the bootstrap server MUST be signed for it to be of any use to the device.

How devices process redirect information is described more formally in Section 7.5.

2.2. Onboarding Information

Bootstrap information provides all the data necessary for a device to bootstrap itself, in order to be considered ready to be managed (e.g., by an NMS). As defined in this document, this data includes information about a boot image the device **MUST** be running, an initial configuration the device **MUST** commit, and optional scripts that, if specified, the device **MUST** successfully execute.

Bootstrap information is YANG modeled data formally defined by the "onboarding-information" container in the YANG module presented in Section 8.3. This container has the tree diagram shown below. Please see Section 1.4 for tree diagram notation.

```

+---:(onboarding-information)
  +--ro onboarding-information
    +--ro boot-image
      |   +--ro name          string
      |   +--ro (hash-algorithm)
      |   |   +---:(sha256)
      |   |   |   +--ro sha256?    string
      |   +--ro uri*         inet:uri
    +--ro configuration-handling      enumeration
    +--ro pre-configuration-script?   script
    +--ro configuration?
    +--ro post-configuration-script?  script

```

Bootstrap information **MUST** be trusted for it to be of any use to a device. There is no option for a device to process untrusted onboarding information.

Bootstrap information is trusted whenever it is obtained via a secure connection to a trusted bootstrap server, or whenever it is signed by the device's owner. In all other cases, the onboarding information is untrusted.

How devices process onboarding information is described more formally in Section 7.6.

3. Artifacts

This document defines the following three artifacts that can be made available to devices while they are bootstrapping. As will be seen in Section 5, each source of bootstrapping information specifies a means for providing each of the artifacts defined in this section.

3.1. Zero Touch Information

The zero touch information artifact encodes the essential bootstrapping data for the device. This artifact is used to encode the redirect information and onboarding information types discussed in Section 2.

The zero touch information artifact is a PKCS#7 SignedData structure, as specified by Section 9.1 of [RFC2315], encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690. The PKCS#7 structure MUST contain JSON-encoded content conforming to the YANG module specified in Section 8.3.

In order for the zero touch information artifact to be trusted when conveyed over an untrusted transport, the PKCS#7 structure MUST also contain a 'signerInfo' structure, as described in Section 9.1 of [RFC2315], containing a signature generated over the content using the private key associated with the owner certificate (Section 3.2).

3.2. Owner Certificate

The owner certificate artifact is a certificate that is used to identify an 'owner' (e.g., an organization), as known to a trusted certificate authority. The owner certificate is signed by a trusted certificate authority (CA), whose certificate is placed into the ownership voucher (Section 3.3).

The owner certificate is used by a device to verify the signature attached to the zero touch information artifact (Section 3.1) that the device SHOULD have also received, as described in Section 4. In particular, the device verifies signature using the public key in the owner certificate over the content contained within the zero touch information artifact.

In order to validate the owner certificate, a device MUST verify that the owner certificate's certificate-chain includes the certificate specified by the ownership voucher (Section 3.3) that the device SHOULD have also received, as described in Section 4, and the device MUST verify that owner certificate contains an identifier matching the one specified in the voucher and, for devices that verify certificate revocation status, the device MUST also verify that the certificate has neither expired nor been revoked.

The owner certificate artifact is formally an unsigned PKCS #7 SignedData structure as specified by Section 9.1 in [RFC2315], encoded using ASN.1 distinguished encoding rules (DER), as specified in ITU-T X.690.

The owner certificate PKCS#7 structure MUST contain the owner certificate itself, as well as all intermediate certificates leading up to the trust anchor certificate specified in the ownership voucher. The owner certificate artifact MAY optionally include the trust anchor certificate.

Additionally, in order to support devices deployed on private networks, the owner certificate PKCS#7 structure MAY also contain suitably fresh CRLs [RFC5280] and/or OCSP Responses [RFC6960]. Having these revocation objects stapled to the owner certificate precludes the need for the device to have to download them dynamically using the CRL distribution point or an OCSP responder specified in the associated certificates.

3.3. Ownership Voucher

The ownership voucher artifact is used to securely identify a device's owner, as it is known to the manufacturer. The ownership voucher is signed by the device's manufacturer or delegate.

More specifically, the ownership voucher is used to verify the owner certificate (Section 3.2) that the device SHOULD have also received, as described in Section 4. In particular, the device verifies that the owner certificate has a chain of trust leading to the trusted certificate included in the ownership voucher, even if it is itself (e.g., self-signed certificate).

In order to validate the ownership voucher, a device MUST perform a number of checks. The device MUST verify that the voucher specifies the device's serial number. The device MUST verify that the ownership voucher has a chain of trust to a trusted certificate known to the device (Section 7.1). If the ownership voucher contains an expiration date, the device MUST also verify that the ownership voucher has not expired.

The ownership voucher artifact, including its encoding, is formally defined in [I-D.ietf-anima-voucher].

4. Artifact Groupings

Section 3 lists all the possible bootstrapping artifacts, but only certain groupings of these artifacts make sense to return in the various bootstrapping situations described in this document. The remainder of this section identifies these groupings to further clarify how the artifacts are used.

4.1. Unsigned Information

The first grouping of artifacts is for unsigned information. That is, when the zero touch information artifact (Section 3.1) has not been signed.

Unsigned information is useful for cases when transport level security can be used to convey trust (e.g., HTTPS), or when the information can be processed in a provisional manner (i.e. unsigned redirect information).

Conveying unsigned information entails communicating just one of the three artifacts listed in Section 3 as follows:

List of artifacts included in this grouping:

- zero touch information (with no embedded signature)

4.2. Signed Information (without Revocations)

The second grouping of artifacts is for when the zero touch information artifact (Section 3.1) has been signed, but without any revocation information, because the device is expected to download the revocation information dynamically (e.g., using the CRL distribution point or OCSP Responder listed in the owner certificate and the pinned domain certificate specified in the ownership voucher).

Signed information is needed when the information is obtained from an untrusted source of bootstrapping data (Section 5), in order for the device to be able to trust the information.

Revocation information may not need to be provided because, for instance, the device only uses revocation information obtained dynamically from Internet based resources. Another possible reason may be because the device does not have a reliable clock, and therefore the manufacturer decides to never revoke information (e.g., ownership assignments are forever).

Conveying signed information without revocation information entails communicating all three of the artifacts listed in Section 3 as follows:

List of artifacts included in this grouping:

- zero touch information (with an embedded signature)
- owner certificate (with no stapled revocation objects)
- ownership voucher (with no stapled revocation objects)

4.3. Signed Information (with Revocations)

The third grouping of artifacts is for when the zero touch information artifact (Section 3.1) has been signed and also includes revocation information.

Signed information, as described above, is needed when the information is obtained from an untrusted source of bootstrapping data (Section 5), in order for the device be able to trust the information.

Revocation information may need to be provided because, for instance, the device is deployed on a private network and therefore unable to obtain the revocation information from Internet based resources.

Conveying signed information with revocation information entails communicating all three of the artifacts listed in Section 3 as follows:

List of artifacts included in this grouping:

- zero touch information (with an embedded signature)
- owner certificate (with stapled revocation objects)
- ownership voucher (with stapled revocation objects)

5. Sources of Bootstrapping Data

This section defines some sources for zero touch bootstrapping data that a device can access. The list of sources defined here is not meant to be exhaustive. It is left to future documents to define additional sources for obtaining zero touch bootstrapping data.

For each source defined in this section, details are given for how each of the three artifacts listed in Section 3 is provided.

5.1. Removable Storage

A directly attached removable storage device (e.g., a USB flash drive) MAY be used as a source of zero touch bootstrapping data.

To use a removable storage device as a source of bootstrapping data, a device need only detect if the removable storage device is plugged in and mount its filesystem.

Use of a removable storage device is compelling, as it doesn't require any external infrastructure to work. It is notable that the raw boot image file can be located on the removable storage device, enabling a removable storage device to be a fully self-standing bootstrapping solution.

A removable storage device is an untrusted source of bootstrapping data. This means that the information stored on the removable storage device either **MUST** be signed, or it **MUST** be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a removable storage device presents itself as a filesystem, the bootstrapping artifacts need to be presented as files. The three artifacts defined in Section 3 are mapped to files below.

Artifact to File Mapping:

Zero Touch Information: Mapped to a file containing the binary artifact described in Section 3.1 (e.g., zerotouch-information.pkcs7).

Owner Certificate: Mapped to a file containing the binary artifact described in Section 3.2 (e.g., owner-certificate.pkcs7).

Ownership Voucher: Mapped to a file containing the binary artifact described in Section 3.3 (e.g., ownership-voucher.pkcs7).

The format of the removable storage device's filesystem and the naming of the files are outside the scope of this document. However, in order to facilitate interoperability, it is **RECOMMENDED** devices support open and/or standards based filesystems. It is also **RECOMMENDED** that devices assume a file naming convention that enables more than one instance of bootstrapping data to exist on a removable storage device. The file naming convention **SHOULD** be unique to the manufacturer, in order to enable bootstrapping data from multiple manufacturers to exist on a removable storage device.

5.2. DNS Server

A DNS server **MAY** be used as a source of zero touch bootstrapping data.

Using a DNS server may be a compelling option for deployments having existing DNS infrastructure, as it enables a touchless bootstrapping option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

To use a DNS server as a source of bootstrapping data, a device **MAY** perform a multicast DNS [RFC6762] query searching for the service "_zerotouch._tcp.local.". Alternatively the device **MAY** perform DNS-SD [RFC6763] via normal DNS operation, using the domain returned to

it from the DHCP server; for example, searching for the service "_zerotouch._tcp.example.com".

Unsigned DNS records (e.g., not using DNSSEC as described in [RFC6698]) are an untrusted source of bootstrapping data. This means that the information stored in the DNS records either **MUST** be signed, or it **MUST** be information that can be processed provisionally (e.g., unsigned redirect information).

From an artifact perspective, since a DNS server presents resource records (Section 3.2.1 of [RFC1035]), the bootstrapping artifacts need to be presented as resource records. The three artifacts defined in Section 3 are mapped to resource records below.

Artifact to Resource Record Mapping:

Zero Touch Information: Mapped to a TXT record called "zt-info" containing the base64-encoding of the binary artifact described in Section 3.1.

Owner Certificate: Mapped to a TXT record called "zt-cert" containing the base64-encoding of the binary artifact described in Section 3.2.

Ownership Voucher: Mapped to a TXT record called "zt-voucher" containing the base64-encoding of the binary artifact described in Section 3.3.

TXT records have an upper size limit of 65535 bytes (Section 3.2.1 in RFC1035), since 'RDLENGTH' is a 16-bit field. Please see Section 3.1.3 in RFC4408 for how a TXT record can achieve this size. Due to this size limitation, some zero touch information artifacts may not fit. In particular, onboarding information could hit this upper bound, depending on the size of the included configuration and scripts.

When onboarding information (not redirect information) is provided, it is notable that the URL for the boot-image the device can download would have to point to another server (e.g., http://, ftp://, etc.), as DNS servers do not themselves distribute files.

5.3. DHCP Server

A DHCP server **MAY** be used as a source of zero touch bootstrapping data.

Using a DHCP server may be a compelling option for deployments having existing DHCP infrastructure, as it enables a touchless bootstrapping

option that does not entail utilizing an Internet based resource hosted by a 3rd-party.

A DHCP server is an untrusted source of bootstrapping data. Thus the information stored on the DHCP server either MUST be signed, or it MUST be information that can be processed provisionally (e.g., unsigned redirect information).

However, unlike other sources of bootstrapping data described in this document, the DHCP protocol (especially DHCP for IPv4) is limited in the amount of data that can be conveyed, to the extent that signed data cannot be communicated. This means only unsigned redirect information can be conveyed. Since the redirect information is unsigned, it SHOULD NOT include the optional trust anchor certificate, as the device would have to discard it anyway.

From an artifact perspective, the three artifacts defined in Section 3 are mapped to the DHCP fields specified in Section 10 as follows:

Zero Touch Information: This artifact is not supported directly. Instead, the essence of redirect information (not onboarding information) is mapped to the DHCP fields described in Section 10.

Owner Certificate: Not supported. There is not enough space in the DHCP packet to hold an owner certificate artifact.

Ownership Voucher: Not supported. There is not enough space in the DHCP packet to hold an ownership voucher artifact.

5.4. Bootstrap Server

A bootstrap server MAY be used as a source of zero touch bootstrapping data. A bootstrap server is defined as a RESTCONF [RFC8040] server implementing the YANG module provided in Section 9.

Unlike any other source of bootstrap data described in this document, a bootstrap server is not only a source of data, but it can also receive data from devices using the YANG-defined "notification" action statement defined in the YANG module (Section 9.3). The data sent from devices both enables visibility into the bootstrapping process (e.g., warnings and errors) as well as provides potentially useful completion status information (e.g., the device's SSH host-keys).

To use a bootstrap server as a source of bootstrapping data, a device MUST use the RESTCONF protocol to access the YANG container node

/device, passing its own serial number in the URL as the key to the 'device' list.

Using a bootstrap server as a source of bootstrapping data is a compelling option as it MAY use transport-level security, in lieu of signed data, which may be easier to deploy in some situations. Additionally, the bootstrap server is able to receive notifications from devices, which may be critical to some deployments (e.g., the passing of the device's SSH host keys).

A bootstrap server may be trusted or an untrusted source of bootstrapping data, depending on how the device learned about the bootstrap server's trust anchor from a trusted source. When a bootstrap server is trusted, the information returned from it MAY be signed. However, when the server is untrusted, in order for its information to be of any use to the device, the bootstrap information MUST either be signed or be information that can be processed provisionally (e.g., unsigned redirect information).

When a device is able to trust a bootstrap server, it MUST send its IDevID certificate in the form of a TLS client certificate, and it MUST send notifications to the bootstrap server. When a device is not able to trust a bootstrap server, it MUST NOT send its IDevID certificate in the form of a TLS client certificate, and it MUST NOT send any notifications to the bootstrap server.

From an artifact perspective, since a bootstrap server presents data as a YANG-modeled data, the bootstrapping artifacts need to be mapped to nodes in the YANG module. The three artifacts defined in Section 3 are mapped to bootstrap server nodes defined in Section 9.3 below.

Artifact to Bootstrap Server Node Mapping:

Zero Touch Information: Mapped to the leaf node /device/zerotouch-information.

Owner Certificate: Mapped to the leaf node /device/owner-certificate.

Ownership Voucher: Mapped to the leaf node /device/ownership-voucher.

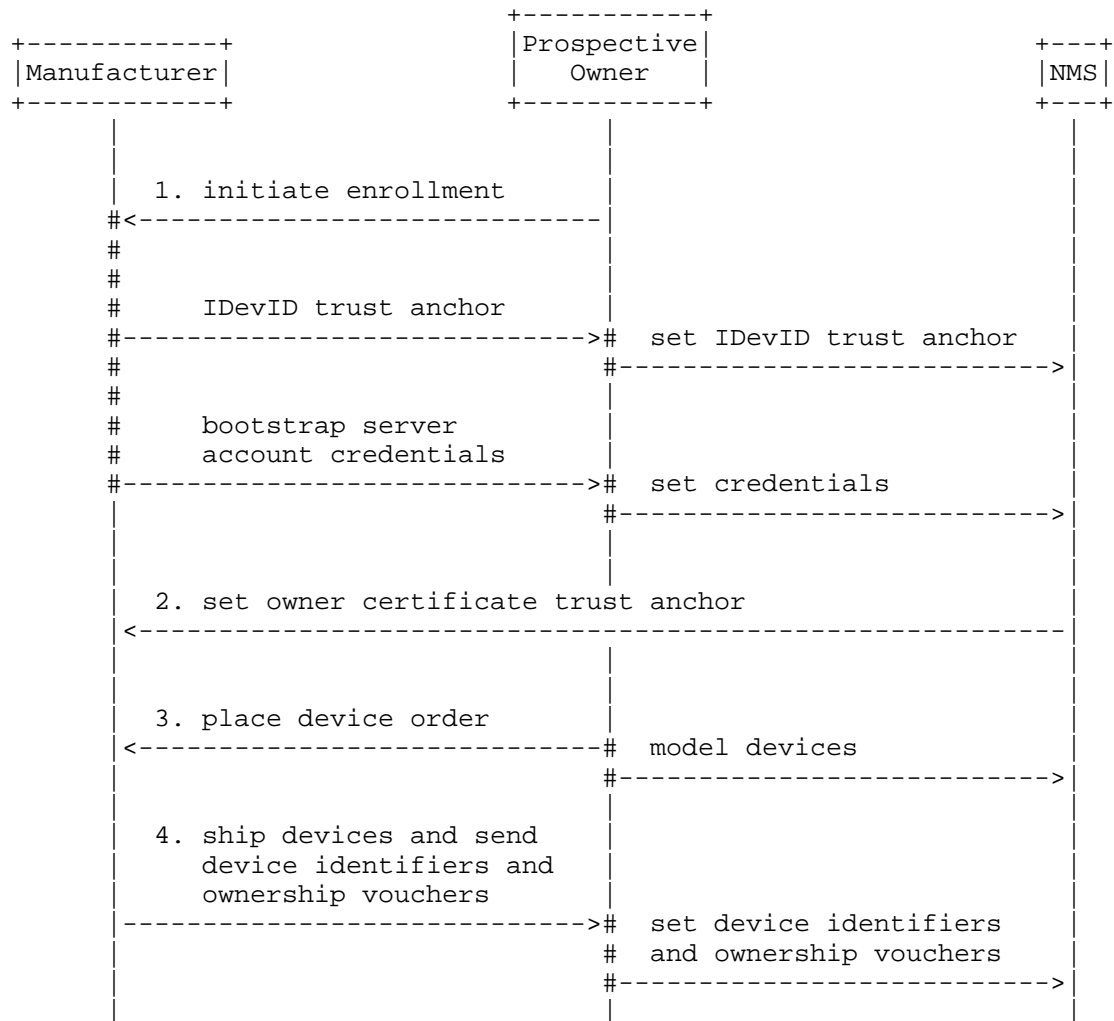
While RESTCONF servers typically support a nested hierarchy of resources, zero touch bootstrap servers only need to support the paths /device and /device/notification. The device processing instructions provided in Section 7.3 only uses these two URLs.

6. Workflow Overview

The zero touch solution presented in this document is conceptualized to be composed of the workflows described in this section. Implementations MAY vary in details. Each diagram is followed by a detailed description of the steps presented in the diagram, with further explanation on how implementations may vary.

6.1. Enrollment and Ordering Devices

The following diagram illustrates key interactions that may occur from when a prospective owner enrolls in a manufacturer's zero touch program to when the manufacturer ships devices for an order placed by the prospective owner.



Each numbered item below corresponds to a numbered item in the diagram above.

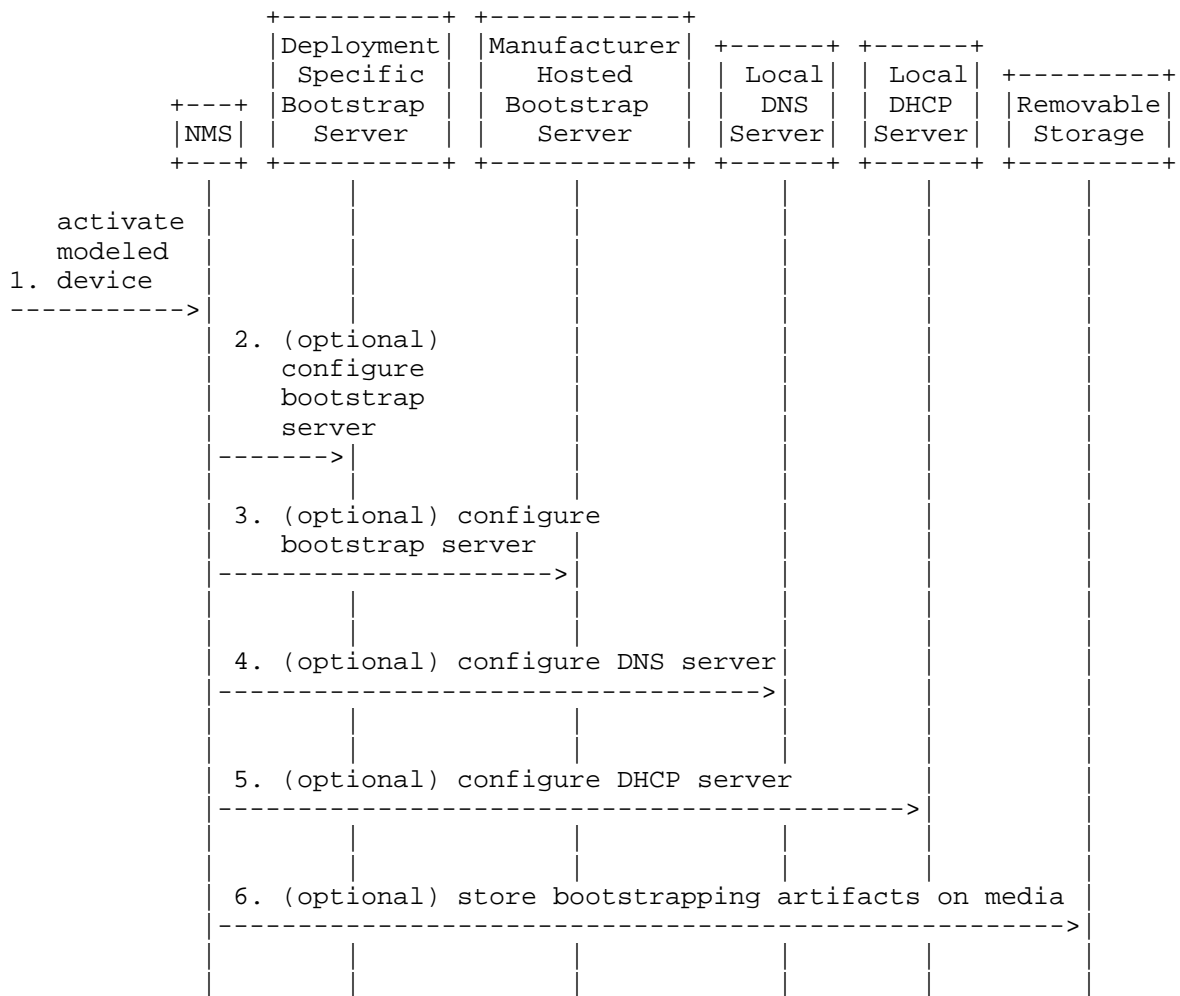
1. A prospective owner of a manufacturer's devices, or an existing owner that wishes to start using zero touch for future device orders, initiates an enrollment process with the manufacturer or delegate. This process includes the following:
 - * Regardless how the prospective owner intends to bootstrap their devices, they will always obtain from the manufacturer or delegate the trust anchor certificate for its device's IDevID certificates. This certificate will need to be

installed on the prospective owner's NMS so that the NMS can subsequently authenticate the devices' IDevID certificates.

- * If the manufacturer hosts an Internet based bootstrap server (e.g., a redirect server) such as described in Section 5.4, then credentials necessary to configure the bootstrap server would be provided to the prospective owner. If the bootstrap server is configurable through an API (outside the scope of this document), then the credentials might be installed on the prospective owner's NMS so that the NMS can subsequently configure the manufacturer-hosted bootstrap server directly.
2. If the manufacturer's devices are able to validate signed data (Section 7.4), and assuming that the prospective owner's NMS is able to prepare and sign the bootstrapping data itself, the prospective owner's NMS might set a trust anchor certificate onto the manufacturer's bootstrap server, using the credentials provided in the previous step. This certificate is the trust anchor certificate that the prospective owner would like the manufacturer to place into the ownership vouchers it generates, thereby enabling devices to trust the owner's owner certificate. How this trust anchor certificate is used to enable devices to validate signed bootstrapping data is described in Section 7.4.
 3. Some time later, the prospective owner places an order with the manufacturer or delegate, perhaps with a special flag checked for zero touch handling. At this time, or perhaps before placing the order, the owner may model the devices in their NMS, creating virtual objects for the devices with no real-world device associations. For instance the model can be used to simulate the device's location in the network and the configuration it should have when fully operational.
 4. When the manufacturer or delegate fulfills the order, shipping the devices to their intended locations, they may notify the owner of the devices' serial numbers and shipping destinations, which the owner may use to stage the network for when the devices power on. Additionally, the manufacturer may send one or more ownership vouchers, cryptographically assigning ownership of those devices to the owner. The owner may set this information on their NMS, perhaps binding specific modeled devices to the serial numbers and ownership vouchers.

6.2. Owner Stages the Network for Bootstrap

The following diagram illustrates how an owner might stage the network for bootstrapping devices.



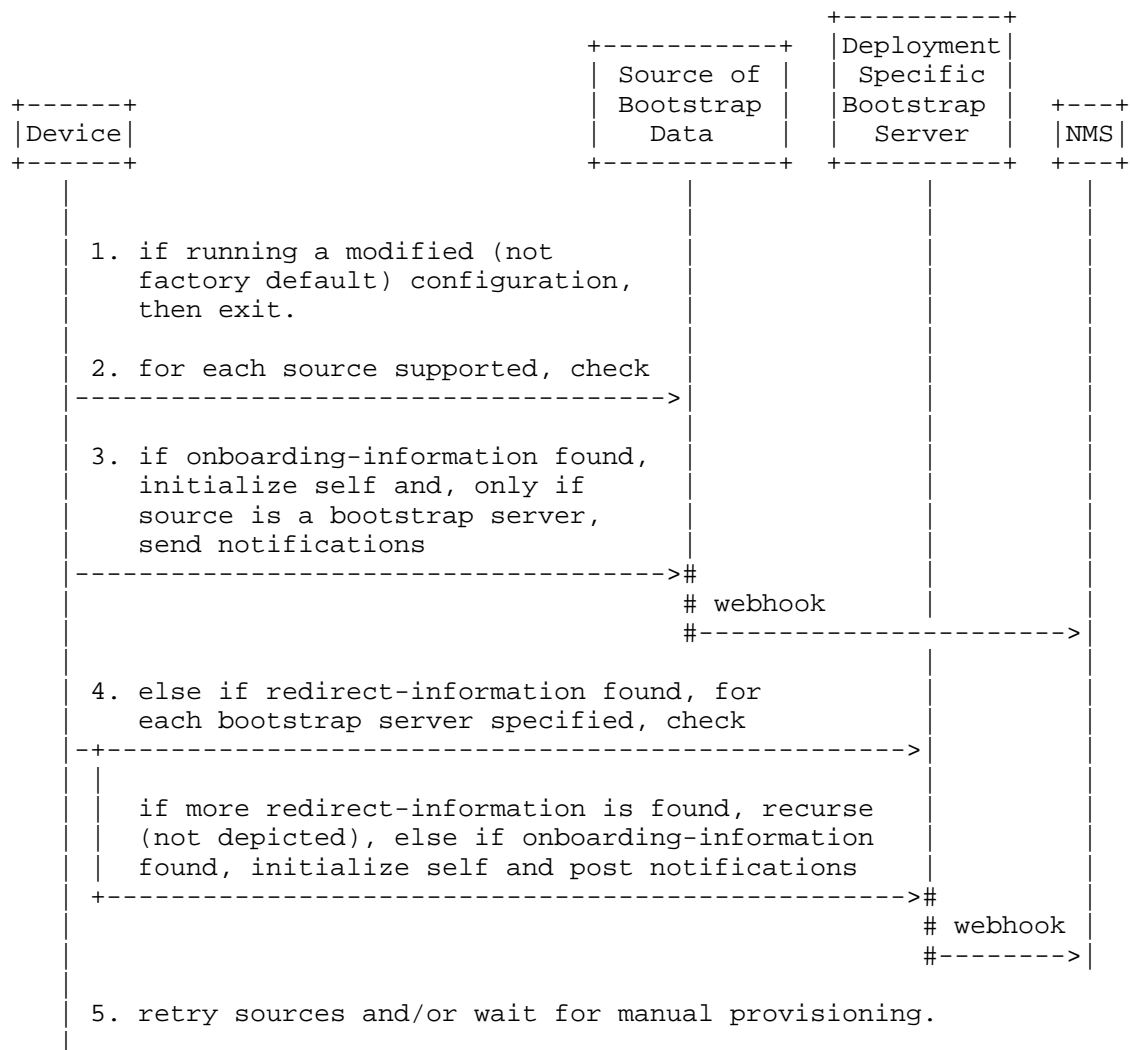
Each numbered item below corresponds to a numbered item in the diagram above.

1. Having previously modeled the devices, including setting their fully operational configurations and associating both device serial numbers and ownership vouchers, the owner might "activate" one or more modeled devices. That is, the owner tells the NMS to perform the steps necessary to prepare for when the real-world devices power up and initiate the bootstrapping process. Note that, in some deployments, this step might be combined with the last step from the previous workflow. Here it is depicted that an NMS performs the steps, but they may be performed manually or through some other mechanism.

2. If it is desired to use a deployment specific bootstrap server, it MUST be configured to provide the bootstrapping information for the specific devices. Configuring the bootstrap server MAY occur via a programmatic API not defined by this document. Illustrated here as an external component, the bootstrap server MAY be implemented as an internal component of the NMS itself.
3. If it is desired to use a manufacturer (or delegate) hosted bootstrap server, it MUST be configured to provide the bootstrapping information for the specific devices. The configuration MUST be either redirect or onboarding information. That is, either the manufacturer hosted bootstrap server will redirect the device to another bootstrap server, or provide the device with its bootstrapping information itself. The types of bootstrapping information the manufacturer hosted bootstrap server supports MAY vary by implementation; some implementations may only support redirect information, or only support onboarding information, or support both redirect and onboarding information. Configuring the bootstrap server MAY occur via a programmatic API not defined by this document.
4. If it is desired to use a DNS server to supply bootstrapping information, a DNS server needs to be configured. If multicast DNS-SD is desired, then the server MUST reside on the local network, otherwise the DNS server MAY reside on a remote network. Please see Section 5.2 for more information about how to configure DNS servers. Configuring the DNS server MAY occur via a programmatic API not defined by this document.
5. If it is desired to use a DHCP server to supply bootstrapping data, a DHCP server needs to be configured. The DHCP server may be accessed directly or via a DHCP relay. Please see Section 5.3 for more information about how to configure DHCP servers. Configuring the DHCP server MAY occur via a programmatic API not defined by this document.
6. If it is desired to use a removable storage device (e.g., USB flash drive) to supply bootstrapping information, the information would need to be placed onto it. Please see Section 5.1 for more information about how to configure a removable storage device.

6.3. Device Powers On

The following diagram illustrates the sequence of activities that occur when a device powers on.



The interactions in the above diagram are described below.

1. Upon power being applied, the device's bootstrapping logic first checks to see if it is running in its factory default state. If it is in a modified state, then the bootstrapping logic exits and none of the following interactions occur.
2. For each source of bootstrapping data the device supports, preferably in order of closeness to the device (e.g., removable storage before Internet based servers), the device checks to see if there is any bootstrapping data for it there.

3. If onboarding information is found, the device initializes itself accordingly (e.g., installing a boot-image and committing an initial configuration). If the source is a bootstrap server, and the bootstrap server can be trusted (i.e., TLS-level authentication), the device also sends progress notifications to the bootstrap server.

- * The contents of the initial configuration SHOULD configure an administrator account on the device (e.g., username, ssh-rsa key, etc.) and SHOULD configure the device either to listen for NETCONF or RESTCONF connections or to initiate call home connections [RFC8071].
- * If the bootstrap server supports forwarding device notifications to external systems (e.g., via a webhook), the "bootstrap-complete" notification (Section 9.3) informs the external system to know when it can, for instance, initiate a connection to the device (assuming it knows the device's address and the device was configured to listen for connections). To support this further, the bootstrap-complete notification MAY also relay the device's SSH host keys and/or TLS certificates, with which the external system can use to authenticate subsequent connections to the device.

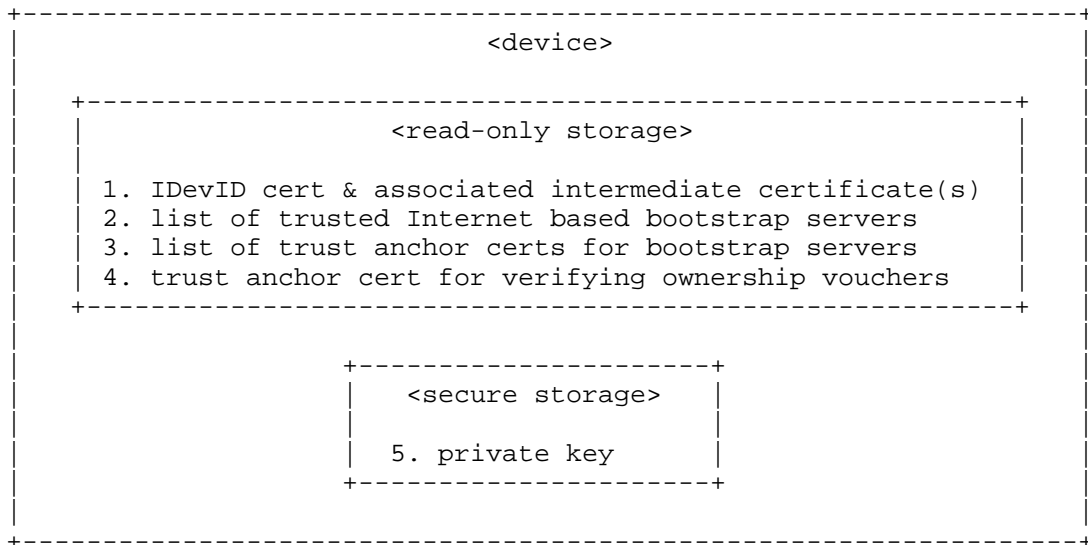
If the device is ever able to complete the bootstrapping process successfully (i.e., no longer running its factory default configuration), it exits the bootstrapping logic without considering any additional sources of bootstrapping data.

4. Otherwise, if redirect information is found, the device iterates through the list of specified bootstrap servers, checking to see if there is any bootstrapping data for it on them. If the bootstrap server returns more redirect information, then the device processes it recursively. Otherwise, if the bootstrap server returns onboarding information, the device processes it following the description provided in (3) above.
5. After having tried all supported sources of bootstrapping data, the device MAY retry again all the sources and/or provide manageability interfaces for manual configuration (e.g., CLI, HTTP, NETCONF, etc.). If manual configuration is allowed, and such configuration is provided, the device MUST immediately cease trying to obtain bootstrapping data, as it would then no longer be in running its factory default configuration.

7. Device Details

Devices supporting the bootstrapping strategy described in this document MUST have the preconfigured factory default state and bootstrapping logic described in the following sections.

7.1. Factory Default State



Each numbered item below corresponds to a numbered item in the diagram above.

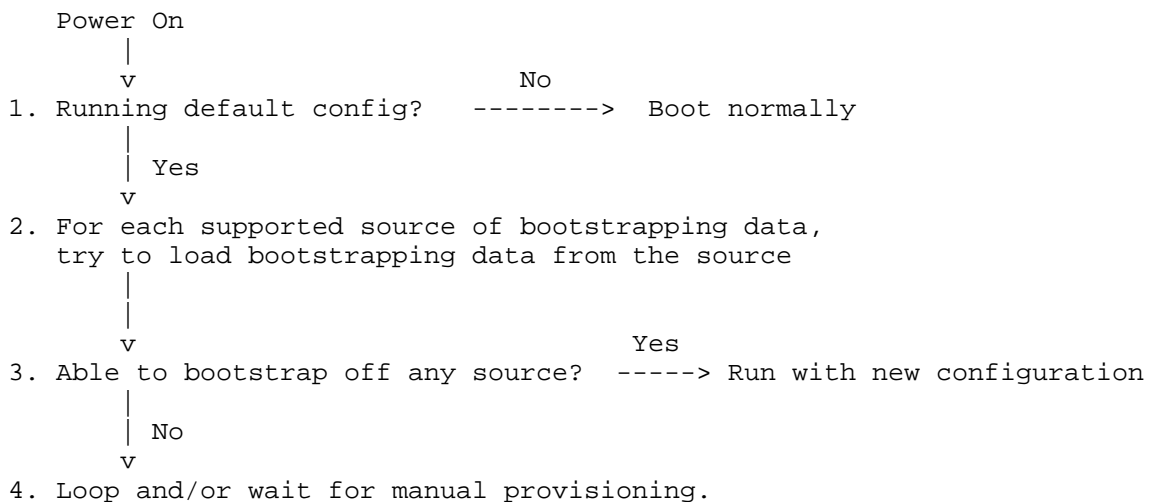
1. Devices MUST be manufactured with an initial device identifier (IDeVID), as defined in [Std-802.1AR-2009]. The IDeVID is an X.509 certificate, encoding the device's serial number. The device MUST also possess any intermediate certificates between the IDeVID certificate and the manufacturer's IDeVID trust anchor certificate, which is provided to prospective owners separately (e.g., Section 6.1).
2. Devices that support loading bootstrapping data from an Internet-based bootstrap server (see Section 5.4) MUST be manufactured with a configured list of trusted bootstrap servers. Consistent with redirect information (Section 2.1, each bootstrap server MAY be identified by its hostname or IP address, and an optional port.
3. Devices that support loading bootstrapping data from an Internet-based bootstrap server (see Section 5.4) MUST also be

manufactured with a list of trust anchor certificates that can be used for X.509 certificate path validation ([RFC6125], Section 6) on the bootstrap server's TLS server certificate.

4. Devices that support loading owner signed data (see Section 1.2) MUST also be manufactured with the manufacturer's trust anchor certificate for the ownership vouchers.
5. Devices MUST be manufactured with a private key that corresponds to the public key encoded in the device's IDevID certificate. This private key SHOULD be securely stored, ideally by a cryptographic processor (e.g., a TPM).

7.2. Boot Sequence

A device claiming to support the bootstrapping strategy defined in this document MUST support the boot sequence described in this section.



Each numbered item below corresponds to a numbered item in the diagram above.

1. When the device powers on, it first checks to see if it is running the factory default configuration. If it is running a modified configuration, then it boots normally.
2. The device iterates over its list of sources for bootstrapping data (Section 5). Details for how to process a source of bootstrapping data are provided in Section 7.3.

3. If the device is able to bootstrap itself off any of the sources of bootstrapping data, it runs with the new bootstrapped configuration.
4. Otherwise the device MAY loop back through the list of bootstrapping sources again and/or wait for manual provisioning.

7.3. Processing a Source of Bootstrapping Data

This section describes a recursive algorithm that devices can use to, ultimately, obtain onboarding information. The algorithm is recursive only because sources of bootstrapping data MAY return redirect information, which causes the algorithm to run again, for the newly discovered sources of information. To be clear, an expression that captures all possible combinations is "(redirect information)* onboarding information". That is, zero or more redirect information responses, followed by one bootstrap information response.

An important aspect of the algorithm is knowing when data needs to be signed or not. The following figure provides a summary of options:

Kind of Bootstrapping Data	Untrusted Source		Trusted Source
	Can Provide?		Can Provide?
Unsigned Redirect Info	:	Yes+	Yes
Signed Redirect Info	:	Yes	Yes*
Unsigned Bootstrap Info	:	No	Yes
Signed Bootstrap Info	:	Yes	Yes*

The '+' above denotes that the source redirected to MUST return signed data, or more unsigned redirect information.

The '*' above denotes that, while possible, it is generally unnecessary for a trusted source to return signed data. In fact, it's only needed when the '+' case occurs.

As an example, imagine a device initially obtains unsigned redirect information, which redirects it to an [untrusted] bootstrap server where it obtains more unsigned redirect information, which redirects it to another [untrusted] bootstrap server where it obtains signed redirect information, which redirects it to a [trusted] bootstrap server where it obtains redirect information (signed or unsigned doesn't matter, its trusted either way), but without an included trust anchor certificate, which is unexpected but possible, so the device can't trust the server it's redirected to, and so on, until finally the device obtains some onboarding information.

To support this behavior, this recursive algorithm uses a conceptually global-scoped variable algorithm variable called "trust-state". The trust-state variable is initialized to FALSE. The ultimate goal of this algorithm is for the device to process onboarding information (Section 2.2) while the trust-state variable is TRUE.

If the data source is a bootstrap server, the only source of bootstrapping data defined in this document that can be trusted via transport level security, and the device is able to authenticate the server using X.509 certificate path validation ([RFC6125], Section 6) to one of the device's preconfigured trust anchors, or to a trust anchor that it learned from a previous step, then the device MUST set trust-state to TRUE.

If trust-state is TRUE, when connecting to the bootstrap server, the device MUST use its IDevID certificate for client certificate based authentication and MUST POST progress notifications using the bootstrap server's "notification" action. Otherwise, if trust-state is FALSE, when connecting to the bootstrap server, the device MUST NOT use its IDevID certificate for a client certificate based authentication and MUST NOT POST progress notifications using the bootstrap server's "notification" action.

When accessing a bootstrap server, the device SHOULD only access its top-level resource, to obtain all the data staged for it in a single GET request.

For any source of bootstrapping data (e.g., Section 5), if the data is signed and the device is able to validate the signed data using the algorithm described in Section 7.4, then the device MUST set trust-state to TRUE, else the device MUST set trust-state to FALSE. Note, this is worded to cover the special case when signed data is returned even from a trusted bootstrap server.

If the data is onboarding information (not redirect information), and trust-state is FALSE, the device MUST exit the recursive algorithm (as this is not allowed, per the figure above), returning to the state machine described in Section 7.2. Otherwise, the device MUST attempt to process the onboarding information as described in Section 7.6. In either case, success or failure, the device MUST exit the recursive algorithm, returning to the state machine described in Section 7.2, the only difference being in how it responds to the "Able to bootstrap off any source?" conditional described in the figure in the section.

If the data is redirect information, the device MUST process the redirect information as described in Section 7.5. This is the

recursion step, it will cause to device to reenter this algorithm, but this time the data source will most definitely be a bootstrap server, as that is all redirect information is able to redirect a device to.

7.4. Validating Signed Data

Whenever a device is presented signed data from an untrusted source, it MUST validate the signed data as described in this section. If the signed data is provided by a trusted source, a redundant trust case, the device MAY skip verifying the signature.

Whenever there is signed data, the device MUST also be provided an ownership voucher and an owner certificate. Depending on circumstances, the device MAY also be provided certificate revocations. How all the needed artifacts are provided for each source of bootstrapping data is defined in Section 5.

The device MUST first authenticate the ownership voucher by validating the signature on it to one of its preconfigured trust anchors (see Section 7.1) and verify that the ownership voucher contains the device's serial number. If the ownership voucher contains an expiration timestamp, the device MUST also verify that the ownership voucher has not expired. If the authentication of the ownership voucher is successful, the device extracts from it information that can be used to verify the owner certificate in the next step.

Next the device MUST authenticate the owner certificate by performing X.509 certificate path verification to the trusted certificate provided in the voucher. If the device insists on verifying revocation status, it MUST also verify that none of the certificates in the chain of certificates have been revoked or expired. If the authentication of the owner certificate is successful, the device extracts the owner's public key from the owner certificate for use in the next step.

Finally the device MUST verify the signature over information artifact was generated by the private key matching the public key extracted from the owner certificate in the previous step.

If any of these steps fail, then the device MUST mark the data as invalid and not perform any of the subsequent steps.

7.5. Processing Redirect Information

In order to process redirect information (Section 2.1), the device MUST follow the steps presented in this section.

Processing redirect information is straightforward. The device sequentially steps through the list of provided bootstrap servers until it can find one it can bootstrap off of.

If a hostname is provided, and the hostname's DNS resolution is to more than one IP address, the device MUST attempt to connect to all of the DNS resolved addresses at least once, before moving on to the next bootstrap server. If the device is able to obtain bootstrapping data from any of the DNS resolved addresses, it MUST immediately process that data, without attempting to connect to any of the other DNS resolved addresses.

If the redirect information is trusted (e.g., trust-state is TRUE), and the bootstrap server entry contains a trust anchor certificate, then the device MUST authenticate the bootstrap server using X.509 certificate path validation ([RFC6125], Section 6) to the specified trust anchor. If the device is unable to authenticate the bootstrap server to the specified trust anchor, the device MUST NOT attempt a provisional connection to the bootstrap server (i.e., by blindly accepting its server certificate).

If the redirect information is untrusted (e.g., trust-state is FALSE), the device MUST discard any trust anchors provided by the redirect information and establish a provisional connection to the bootstrap server (i.e., by blindly accepting its TLS server certificate).

7.6. Processing Onboarding Information

In order to process onboarding information (Section 2.2), the device MUST follow the steps presented in this section.

When processing onboarding information, the device MUST first process the boot image information, then execute the pre-configuration script (if any), then commit the initial configuration, and then execute the script (if any), in that order. If the device encounters an error at any step, it MUST NOT proceed to the next step.

First the device MUST determine if the image it is running satisfies the specified boot image criteria (e.g., name or fingerprint match). If it does not, the device MUST download (using the supplied URI), verify, and install the specified boot image, and then reboot. To verify the boot image, the device MUST check that the boot image file

matches the fingerprint (e.g., sha256) supplied by the bootstrapping information. Upon rebooting, the device MUST still be in its factory default state, causing the bootstrapping process to run again, which will eventually come to this very point, but this time the device's running image will satisfy the specified criteria, and thus the device will move to processing the next step.

Next, for devices that support executing scripts, if a pre-configuration script has been specified, the device MUST execute the script and check its exit status code to determine if it had any warnings or errors. In the case of errors, the device MUST reset itself in such a way that force the reinstallation of its boot image, thereby wiping out any bad state the script might have left behind.

Next the device commits the provided initial configuration. Assuming no errors, the device moves to processing the next step.

Again, for devices that support executing scripts, if a post-configuration script has been specified, the device MUST execute the script and check its exit status code to determine if it had any warnings or errors. In the case of errors, the device MUST reset itself in such a way that force the reinstallation of its boot image, thereby wiping out any bad state the script might have left behind.

At this point, the device has completely processed the bootstrapping data and is ready to be managed. If the device obtained the bootstrap information from a trusted bootstrap server, the device MUST send the 'bootstrap-complete' notification now.

At this point the device is configured and no longer running its factory default configuration. Notably, if the onboarding information configured the device it initiate a call home connection, the device would proceed to do so now.

8. The Zero Touch Information Artifact

This section defines a YANG [RFC6020] module that is used to define the data model for the zero touch information artifact described in Section 3.1. Examples illustrating this artifact in use are provided in Section 8.2.

8.1. Tree Diagram

The following tree diagram provides an overview of the data model for the zero touch information artifact. The syntax used for this tree diagram is described in Section 1.4.

```

module: ietf-zerotouch-information
+---- (information-type)
+--:(redirect-information)
|   +---- redirect-information
|   |   +---- bootstrap-server* [address]
|   |   |   +---- address      inet:host
|   |   |   +---- port?       inet:port-number
|   |   |   +---- trust-anchor? binary
|   +--:(onboarding-information)
|   |   +---- onboarding-information
|   |   |   +---- boot-image
|   |   |   |   +---- name      string
|   |   |   |   +---- (hash-algorithm)
|   |   |   |   |   +--:(sha256)
|   |   |   |   |   |   +---- sha256? string
|   |   |   |   +---- uri*      inet:uri
|   |   +---- configuration-handling? enumeration
|   |   +---- pre-configuration-script? script
|   |   +---- configuration?          <anydata>
|   |   +---- post-configuration-script? script

```

8.2. Example Usage

This section presents examples for how the zero touch information artifact (Section 3.1) can be encoded into a document that can be distributed outside the bootstrap server's RESTCONF API.

The following example illustrates how redirect information can be encoded into an artifact.

```

<redirect-information
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-information">
  <bootstrap-server>
    <address>phs1.example.com</address>
    <port>8443</port>
    <trust-anchor>Base64-encoded X.50 </trust-anchor>
  </bootstrap-server>
  <bootstrap-server>
    <address>phs2.example.com</address>
    <port>8443</port>
    <trust-anchor>Base64-encoded X.50 </trust-anchor>
  </bootstrap-server>
  <bootstrap-server>
    <address>phs3.example.com</address>
    <port>8443</port>
    <trust-anchor>Base64-encoded X.50 </trust-anchor>
  </bootstrap-server>
</redirect-information>

```

The following example illustrates how onboarding information can be encoded into an artifact. This example uses data models from [RFC7317] and [I-D.ietf-netconf-netconf-client-server].

<-- '\ ' line wrapping added for formatting purposes only -->

```

<onboarding-information
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zerotouch-information">
  <boot-image>
    <name>boot-image-v3.2R1.6.img</name>
    <sha256>Hex-encoded SHA256 hash</sha256>
    <uri>file:///some/path/to/raw/file </uri>
  </boot-image>
  <configuration-handling>merge</configuration-handling>
  <configuration>
    <!-- from ietf-system.yang -->
    <system xmlns="urn:ietf:params:xml:ns:yang:ietf-system">
      <authentication>
        <user>
          <name>admin</name>
          <authorized-key>
            <name>admin's rsa ssh host-key</name>
            <algorithm>ssh-rsa</algorithm>
            <key-data>AAAAB3NzaC1yc2EAAAADAQABAAQDeJMV8zrtsi8CgEsRC\
jCzfve2m6zD3awSBPrh7ICgQLQvHVbPL89eHLuecStKL3HrEgXaI/O2Mwj\
E1lG9YxLzeS5p2ngzK6lvikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVcC\
WAw1lOr9IDGAuww6G45gLCHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA5\
vg7SLqQFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jWq\

```

```

        EIUa7LvEJYql4unq4Iog+/+CiumTkmQIWRgIoJ4FCzYkO9NvRE6fOSLLf6\
        gakWVOZZgQ8929uWjCWlGlqn2mPibp2Go1</key-data>
    </authorized-key>
</user>
</authentication>
</system>
<!-- from ietf-netconf-server.yang -->
<netconf-server
  xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <call-home>
    <netconf-client>
      <name>config-mgr</name>
      <ssh>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <address>11.22.33.44</address>
          </endpoint>
          <endpoint>
            <name>west-data-center</name>
            <address>55.66.77.88</address>
          </endpoint>
        </endpoints>
        <host-keys>
          <host-key>
            <name>certificate</name>
            <certificate>builtin-idevid-cert</certificate>
          </host-key>
        </host-keys>
        <client-cert-auth>
          <trusted-ca-certs>deployment-specific-ca-certs</trusted-ca-certs>
          <trusted-client-certs>explicitly-trusted-client-certs</trusted-clie
ent-certs>
        </client-cert-auth>
      </ssh>
    <connection-type>
      <periodic>
        <idle-timeout>300</idle-timeout>
        <reconnect-timeout>60</reconnect-timeout>
      </periodic>
    </connection-type>
    <reconnect-strategy>
      <start-with>last-connected</start-with>
      <max-attempts>3</max-attempts>
    </reconnect-strategy>
  </netconf-client>
</call-home>
</netconf-server>
</configuration>

```

</onboarding-information>

8.3. YANG Module

The zero touch information artifact is normatively defined by the YANG module defined in this section.

Note: the module defined herein uses data types defined in [RFC5280], [RFC6234], and [RFC6991].

<CODE BEGINS> file "ietf-zerotouch-information@2017-06-19.yang"

```
module ietf-zerotouch-information {
  yang-version "1.1";

  namespace "urn:ietf:params:xml:ns:yang:ietf-zerotouch-information";
  prefix "zti";

  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types";
  }

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is only present to access
       the yang-data extension defined in RFC 8040.";
    reference "RFC 8040: RESTCONF Protocol";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:  http://tools.ietf.org/wg/netconf
     WG List:  <mailto:netconf@ietf.org>
     Author:   Kent Watsen <mailto:kwatsen@juniper.net>";

  description
    "This module defines the data model for the Zero Touch Information
     artifact defined by RFC XXXX: Zero Touch Provisioning for NETCONF
     or RESTCONF based Management.

     Copyright (c) 2014 IETF Trust and the persons identified as
     authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with or without
```

modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

```
revision "2017-06-19" {
  description
    "Initial version";
  reference
    "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF based
    Management";
}

rc:yang-data zerotouch-information {

  choice information-type {
    mandatory true;
    description
      "This choice statement ensures the response only contains
      redirect-information or onboarding-information. Note that
      this is the only mandatory true node, as the other nodes
      are not needed when the device trusts the bootstrap server,
      in which case the data does not need to be signed.";

    container redirect-information {
      description
        "This is redirect information, as described in Section 2.1
        in RFC XXXX. Its purpose is to redirect a device to another
        bootstrap server.";
      reference
        "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF
        based Management";

      list bootstrap-server {
        key address;
        description
          "A bootstrap server entry.";

        leaf address {
          type inet:host;
          mandatory true;
          description
            "The IP address or hostname of the bootstrap server the
            device should redirect to.";
        }
      }
    }
  }
}
```

```
leaf port {
  type inet:port-number;
  default 443;
  description
    "The port number the bootstrap server listens on.";
}
leaf trust-anchor { //should there be two fields like voucher?
  type binary;
  description
    "An X.509 v3 certificate structure as specified by RFC
    5280, Section 4, encoded using ASN.1 distinguished
    encoding rules (DER), as specified in ITU-T X.690. A
    certificate that a device can use as a trust anchor
    to authenticate the bootstrap server it is being
    redirected to.";
  reference
    "RFC 5280:
      Internet X.509 Public Key Infrastructure Certificate
      and Certificate Revocation List (CRL) Profile.
    ITU-T X.690:
      Information technology - ASN.1 encoding rules:
      Specification of Basic Encoding Rules (BER),
      Canonical Encoding Rules (CER) and Distinguished
      Encoding Rules (DER).";
}
}
}

container onboarding-information {

  description
    "This is bootstrap information, as described in Section 2.2 in
    RFC XXXX. Its purpose is to provide the device everything it
    needs to bootstrap itself.";
  reference
    "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF
    based Management";

  container boot-image {
    description
      "Specifies criteria for the boot image the device MUST
      be running.";

    leaf name { // maybe this should be a regex?
      type string;
      mandatory true;
      description
        "The name of a software image that the device MUST
```

```
        be running in order to process the remaining nodes.";
    }
    choice hash-algorithm {
        mandatory true;
        description
            "Identifies the hash algorithm used.";
        leaf sha256 {
            type string;
            description
                "The hex-encoded SHA-256 hash over the boot
                 image file. This is used by the device to
                 verify a downloaded boot image file.";
            reference
                "RFC 6234: US Secure Hash Algorithms.";
        }
    }
    leaf-list uri {
        type inet:uri;
        min-elements 1;
        description
            "An ordered list of URIs to where the boot-image file MAY
             be obtained. Deployments MUST know in which URI schemes
             (http, ftp, etc.) a device supports. If a secure scheme
             (e.g., https) is provided, a device MAY establish a
             provisional connection to the server, by blindly
             accepting the server's credentials (e.g., its TLS
             certificate)";
    }
}

leaf configuration-handling {
    type enumeration {
        enum merge {
            description
                "Merge configuration into existing running configuration.";
        }
        enum replace {
            description
                "Replace existing running configuration with the passed
                 configuration.";
        }
    }
    description
        "This enumeration indicates how the server should process
         the provided configuration. When not specified, the device
         MAY determine how to process the configuration using other
         means (e.g., vendor-specific metadata).";
}
```



```

    leaf pre-configuration-script {
        type script;
        description
            "A script that, when present, is executed before the
            configuration has been processed.";
    }

    anydata configuration {
        must "../configuration-handling";
        description
            "Any configuration data model known to the device. It may
            contain manufacturer-specific and/or standards-based data
            models.";
    }

    leaf post-configuration-script {
        type script;
        description
            "A script that, when present, is executed after the
            configuration has been processed.";
    }
}
}
}

```

```

typedef script {
    type binary;
    description
        "A device specific script that enables the execution of commands
        to perform actions not possible thru configuration alone."

```

No attempt is made to standardize the contents, running context, or programming language of the script. The contents of the script are considered specific to the vendor, product line, and/or model of the device.

If a script is erroneously provided to a device that does not support the execution of scripts, the device SHOULD send a 'script-warning' notification message, but otherwise continue processing the bootstrapping data as if the script had not been present.

The script returns exit status code '0' on success and non-zero on error, with accompanying stderr/stdout for logging purposes. In the case of an error, the exit status code will specify what the device should do.

If the exit status code is greater than zero, then the device

should assume that the script had a soft error, which the script believes does not affect manageability. If the device obtained the bootstrap information from a bootstrap server, it SHOULD send a 'script-warning' notification message.

If the exit status code is less than zero, the device should assume the script had a hard error, which the script believes will affect manageability. In this case, the device SHOULD send a 'script-error' notification message followed by a reset that will force a new boot-image install (wiping out anything the script may have done) and restart the entire bootstrapping process again.";

}

}

<CODE ENDS>

9. The Zero Touch Bootstrap Server API

This section defines a YANG [RFC6020] module that is used to define the RESTCONF [RFC8040] API used by the bootstrap server defined in Section 5.4. Examples illustrating this API in use are provided in Section 9.2.

9.1. Tree Diagram

The following tree diagram provides an overview for the bootstrap server RESTCONF API. The syntax used for this tree diagram is described in Section 1.4.

```

module: ietf-zerotouch-bootstrap-server
  +--ro device* [unique-id]
    +--ro unique-id          string
    +--ro zerotouch-information pkcs7
    +--ro owner-certificate?  pkcs7
    +--ro ownership-voucher?  pkcs7
    +---x notification
      +---w input
        +---w notification-type enumeration
        +---w message?         string
        +---w ssh-host-keys
          +---w ssh-host-key*
            +---w format        enumeration
            +---w key-data      string
        +---w trust-anchors
          +---w trust-anchor*
            +---w protocol*     enumeration
            +---w certificate    pkcs7

```

In the above diagram, notice that all of the protocol accessible nodes are read-only, to assert that devices can only pull data from the bootstrap server.

Also notice that the module defines an action statement, which devices use to provide progress notifications to the bootstrap server.

9.2. Example Usage

This section presents some examples illustrating the bootstrap server's API. Two examples are provided, one illustrating a device fetching bootstrapping data from the server, and the other illustrating a data posting a progress notification to the server.

The following example illustrates a device using the API to fetch its bootstrapping data from the bootstrap server. In this example, the device receives a signed response; an unsigned response would look similar except the last two fields (owner-certificate and ownership-voucher) would be absent in the response.

REQUEST

['\ ' line wrapping added for formatting only]

```
GET https://example.com/restconf/data/ietf-zero-touch-bootstrap-server:\
device=123456 HTTP/1.1
HOST: example.com
Accept: application/yang.data+xml
```

RESPONSE

```
HTTP/1.1 200 OK
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+xml
```

<!-- '\ ' line wrapping added for formatting purposes only -->

```
<device
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zero-touch-bootstrap-server">
  <unique-id>123456789</unique-id>
  <zero-touch-information>Base64-encoded PKCS#7</zero-touch-information>
  <owner-certificate>Base64-encoded PKCS#7</owner-certificate>
  <ownership-voucher>Base64-encoded PKCS#7</ownership-voucher>
</device>
```

The following example illustrates a device using the API to post a notification to a bootstrap server. Illustrated below is the 'bootstrap-complete' message, but the device may send other notifications to the server while bootstrapping (e.g., to provide status updates). In this message, the device is sending both its SSH host keys and TLS server certificate, which the bootstrap server may, for example, pass to an NMS, as discussed in Section 6.3.

Note that devices that are able to present an IDevID certificate [Std-802.1AR-2009] when establishing SSH or TLS connections do not need to include its DevID certificate in the bootstrap-complete message. It is unnecessary to send the DevID certificate in this case because the IDevID certificate does not need to be pinned by an NMS in order to be trusted.

Note that the bootstrap server MUST NOT process a notification from a device without first authenticating the device. This is in contrast to when a device is fetching data from the server, a read-only operation, in which case device authentication is not strictly required (e.g., when sending signed information).

REQUEST

['\`' line wrapping added for formatting only]

POST https://example.com/restconf/data/ietf-zerothouch:\
device=123456/notification HTTP/1.1
HOST: example.com
Content-Type: application/yang.data+xml

<!-- '\`' line wrapping added for formatting purposes only -->

```
<input
  xmlns="urn:ietf:params:xml:ns:yang:ietf-zerothouch-bootstrap-server">
  <notification-type>bootstrap-complete</notification-type>
  <message>example message</message>
  <ssh-host-keys>
    <ssh-host-key>
      <format>ssh-rsa</format>
      <key-data>Base64-encoded SSH RSA Public Key</key-data>
    </ssh-host-key>
    <ssh-host-key>
      <format>ssh-dsa</format>
      <key-data>Base64-encoded SSH DSA Public Key</key-data>
    </ssh-host-key>
  </ssh-host-keys>
  <trust-anchors>
    <trust-anchor>
      <protocol>netconf-ssh</protocol>
      <protocol>netconf-tls</protocol>
      <protocol>restconf-tls</protocol>
      <protocol>netconf-ch-ssh</protocol>
      <protocol>netconf-ch-tls</protocol>
      <protocol>restconf-ch-tls</protocol>
      <certificate>Base64-encoded X.509</certificate>
    </trust-anchor>
  </trust-anchors>
</input>
```

RESPONSE

HTTP/1.1 204 No Content
Date: Sat, 31 Oct 2015 17:02:40 GMT
Server: example-server

9.3. YANG Module

The bootstrap server's device-facing API is normatively defined by the YANG module defined in this section.

Note: the module defined herein uses data types defined in [RFC2315], [RFC5280], and [I-D.ietf-anima-voucher].

<CODE BEGINS> file "ietf-zerotouch-bootstrap-server@2017-06-19.yang"

```
module ietf-zerotouch-bootstrap-server {
  yang-version "1.1";

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server";
  prefix      "ztbs";

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:    <http://tools.ietf.org/wg/netconf/>
    WG List:    <mailto:netconf@ietf.org>
    Author:     Kent Watsen
                <mailto:kwatsen@juniper.net>";

  description
    "This module defines an interface for bootstrap servers, as defined
    by RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF based
    Management.

    Copyright (c) 2014 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or without
    modification, is permitted pursuant to, and subject to the license
    terms contained in, the Simplified BSD License set forth in Section
    4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX; see the RFC
    itself for full legal notices.";

  revision "2017-06-19" {
    description
      "Initial version";
    reference
      "RFC XXXX: Zero Touch Provisioning for NETCONF or RESTCONF based
```

```
    Management";
}

// typedefs

typedef pkcs7 {
    type binary;
    description
        "A PKCS #7 SignedData structure, as specified by Section 9.1
        in RFC 2315, encoded using ASN.1 distinguished encoding rules
        (DER), as specified in ITU-T X.690.";
    reference
        "RFC 2315:
        PKCS #7: Cryptographic Message Syntax Version 1.5.
        ITU-T X.690:
        Information technology - ASN.1 encoding rules:
        Specification of Basic Encoding Rules (BER),
        Canonical Encoding Rules (CER) and Distinguished
        Encoding Rules (DER).";
}

// protocol accessible node

list device {
    key unique-id;
    config false;

    description
        "A device's record entry. This is the only RESTCONF resource
        that a device will GET, as described in Section 8.2 in RFC XXXX.
        Getting just this top-level node provides a device with all the
        data it needs in a single request.";
    reference
        "RFC XXXX: Zero Touch Provisioning for NETCONF or
        RESTCONF based Management";

    leaf unique-id {
        type string;
        description
            "A unique identifier for the device (e.g., serial number).
            Each device accesses its bootstrapping record by its unique
            identifier.";
    }

    leaf zerotouch-information {
        type pkcs7;
        mandatory true;
        description

```

```
"A 'zerotouch-information' artifact, as described in Section
4.1 of RFC XXXX. When conveyed over an untrusted transport, in
order to be processed by a device, this PKCS#7 SignedData
structure MUST contain a 'signerInfo' structure, described
in Section 9.1 of RFC 2315, containing a signature generated
using the owner's private key.";
reference
  "RFC XXXX: Zero Touch Provisioning for NETCONF or
  RESTCONF based Management.
  RFC 2315:
    PKCS #7: Cryptographic Message Syntax Version 1.5";
}

leaf owner-certificate {
  type pkcs7;
  description
    "An unsigned PKCS #7 SignedData structure, as specified by
    Section 9.1 in RFC 2315, encoded using ASN.1 distinguished
    encoding rules (DER), as specified in ITU-T X.690.

    This structure MUST contain the owner certificate and all
    intermediate certificates leading up to at least the trust
    anchor certificate specified in the ownership voucher.
    Additionally, if needed by the device, this structure MAY
    also contain suitably fresh CRL and or OCSP Responses.

    X.509 certificates and CRLs are described in RFC 5280.
    OCSP Responses are described in RFC 6960.";
  reference
    "RFC 2315:
      PKCS #7: Cryptographic Message Syntax Version 1.5.
    RFC 5280:
      Internet X.509 Public Key Infrastructure Certificate
      and Certificate Revocation List (CRL) Profile.
    RFC 6960:
      X.509 Internet Public Key Infrastructure Online
      Certificate Status Protocol - OCSP.
    ITU-T X.690:
      Information technology - ASN.1 encoding rules:
      Specification of Basic Encoding Rules (BER),
      Canonical Encoding Rules (CER) and Distinguished
      Encoding Rules (DER).";
}

leaf ownership-voucher {
  type pkcs7;
  must "../owner-certificate" {
    description
```



```
        "An owner certificate must be present whenever an ownership
        voucher is presented.";
    }
    description
        "A 'voucher' artifact, as described in Section 5 of
        I-D.ietf-anima-voucher. The voucher informs the device
        who it's 'owner' is. The voucher encodes the device's
        serial number, so that the device can be ensured that
        the voucher applies to it. The voucher is signed by
        the device's manufacturer or delagate.";
    reference
        "I-D.ietf-anima-voucher:
        Voucher and Voucher Revocation Profiles for Bootstrapping
        Protocols";
}

action notification {
    input {
        leaf notification-type {
            type enumeration {
                enum bootstrap-initiated {
                    description
                        "Indicates that the device has just accessed the
                        bootstrap server. The 'message' field below MAY
                        contain any additional information that the
                        manufacturer thinks might be useful.";
                }
                enum parsing-warning {
                    description
                        "Indicates that the device had a non-fatal error when
                        parsing the response from the bootstrap server. The
                        'message' field below SHOULD indicate the specific
                        warning that occurred.";
                }
                enum parsing-error {
                    description
                        "Indicates that the device encountered a fatal error
                        when parsing the response from the bootstrap server.
                        For instance, this could be due to malformed encoding,
                        the device expecting signed data when only unsigned
                        data is provided, because the ownership voucher didn't
                        include the device's unique identifier, or because the
                        signature didn't match. The 'message' field below
                        SHOULD indicate the specific error. This notification
                        also indicates that the device has abandoned trying to
                        bootstrap off this bootstrap server.";
                }
                enum boot-image-warning {
```

```
description
    "Indicates that the device encountered a non-fatal
    error condition when trying to install a boot-image.
    A possible reason might include a need to reformat a
    partition causing loss of data. The 'message' field
    below SHOULD indicate any warning messages that were
    generated.";
}
enum boot-image-error {
    description
        "Indicates that the device encountered an error when
        trying to install a boot-image, which could be for
        reasons such as a file server being unreachable,
        file not found, signature mismatch, etc. The
        'message' field SHOULD indicate the specific error
        that occurred. This notification also indicates
        that the device has abandoned trying to bootstrap
        off this bootstrap server.";
}
enum pre-script-warning {
    description
        "Indicates that the device obtained a greater than
        zero exit status code from the script when it was
        executed. The 'message' field below SHOULD indicate
        both the resulting exit status code, as well as
        capture any stdout/stderr messages the script may
        have produced.";
}
enum pre-script-error {
    description
        "Indicates that the device obtained a less than zero
        exit status code from the script when it was executed.
        The 'message' field below SHOULD indicate both the
        resulting exit status code, as well as capture any
        stdout/stderr messages the script may have produced.
        This notification also indicates that the device has
        abandoned trying to bootstrap off this bootstrap
        server.";
}
enum config-warning {
    description
        "Indicates that the device obtained warning messages
        when it committed the initial configuration. The
        'message' field below SHOULD indicate any warning
        messages that were generated.";
}
enum config-error {
    description
```

```
    "Indicates that the device obtained error messages
    when it committed the initial configuration. The
    'message' field below SHOULD indicate the error
    messages that were generated. This notification
    also indicates that the device has abandoned trying
    to bootstrap off this bootstrap server.";
}
enum post-script-warning {
    description
        "Indicates that the device obtained a greater than
        zero exit status code from the script when it was
        executed. The 'message' field below SHOULD indicate
        both the resulting exit status code, as well as
        capture any stdout/stderr messages the script may
        have produced.";
}
enum post-script-error {
    description
        "Indicates that the device obtained a less than zero
        exit status code from the script when it was executed.
        The 'message' field below SHOULD indicate both the
        resulting exit status code, as well as capture any
        stdout/stderr messages the script may have produced.
        This notification also indicates that the device has
        abandoned trying to bootstrap off this bootstrap
        server.";
}
enum bootstrap-complete {
    description
        "Indicates that the device successfully processed the
        all the bootstrapping data and that it is ready to be
        managed. The 'message' field below MAY contain any
        additional information that the manufacturer thinks
        might be useful. After sending this notification,
        the device is not expected to access the bootstrap
        server again.";
}
enum informational {
    description
        "Indicates any additional information not captured by
        any of the other notification-type. For instance, a
        message indicating that the device is about to reboot
        after having installed a boot-image could be provided.
        The 'message' field below SHOULD contain information
        that the manufacturer thinks might be useful.";
}
}
mandatory true;
```

```
    description
      "The type of notification provided.";
  }
  leaf message {
    type string;
    description
      "An optional human-readable value.";
  }
  container ssh-host-keys {
    when "../notification-type = 'bootstrap-complete'" {
      description
        "SSH host keys are only sent when the notification
         type is 'bootstrap-complete'.";
    }
    description
      "A list of SSH host keys an NMS may use to authenticate
       a NETCONF connection to the device with.";
    list ssh-host-key {
      description
        "An SSH host-key";
      leaf format {
        type enumeration {
          enum ssh-dss { description "ssh-dss"; }
          enum ssh-rsa { description "ssh-rsa"; }
        }
        mandatory true;
        description
          "The format of the SSH host key.";
      }
      leaf key-data {
        type string;
        mandatory true;
        description
          "The key data for the SSH host key";
      }
    }
  }
  container trust-anchors {
    when "../notification-type = 'bootstrap-complete'" {
      description
        "Trust anchors are only sent when the notification
         type is 'bootstrap-complete'.";
    }
    description
      "A list of trust anchor certificates an NMS may use to
       authenticate a NETCONF or RESTCONF connection to the
       device with.";
    list trust-anchor {
```

```
description
  "A list of trust anchor certificates an NMS may use to
  authenticate a NETCONF or RESTCONF connection to the
  device with.";
leaf-list protocol {
  type enumeration {
    enum netconf-ssh      { description "netconf-ssh"; }
    enum netconf-tls      { description "netconf-tls"; }
    enum restconf-tls     { description "restconf-tls"; }
    enum netconf-ch-ssh   { description "netconf-ch-ssh"; }
    enum netconf-ch-tls   { description "netconf-ch-tls"; }
    enum restconf-ch-tls  { description "restconf-ch-tls"; }
  }
  min-elements 1;
  description
    "The protocols that this trust anchor secures.";
}
leaf certificate {
  type pkcs7;
  mandatory true;
  description
    "An X.509 v3 certificate structure, as specified by
    Section 4 in RFC5280, encoded using ASN.1 distinguished
    encoding rules (DER), as specified in ITU-T X.690.";
  reference
    "RFC 5280:
    Internet X.509 Public Key Infrastructure Certificate
    and Certificate Revocation List (CRL) Profile.
    ITU-T X.690:
    Information technology - ASN.1 encoding rules:
    Specification of Basic Encoding Rules (BER),
    Canonical Encoding Rules (CER) and Distinguished
    Encoding Rules (DER).";
}
}
}
} // end action
} // end device
}

<CODE ENDS>
```

10. DHCP Zero Touch Options

This section defines two DHCP options, one for DHCPv4 and one for DHCPv6. These two options are semantically the same, though syntactically different.

10.1. DHCPv4 Zero Touch Option

The DHCPv4 Zero Touch Option is used to provision the client with one or more URIs for bootstrap servers that can be contacted to attempt further configuration.

DHCPv4 Zero Touch Redirect Option

```

      0                                     1
      0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+-----+-----+-----+-----+-----+-----+-----+
| option-code (TBD) | option-length |
+-----+-----+-----+-----+-----+-----+
.
. bootstrap-server-list (variable length) .
.
+-----+-----+-----+-----+-----+-----+

```

- o option-code: `OPTION_V4_ZEROTOUCH_REDIRECT` (TBD)
- o option-length: The option length in octets
- o bootstrap-server-list: A list of servers for the client to attempt contacting, in order to obtain further bootstrapping data, in the format shown in [common-field-encoding].

DHCPv4 Client Behavior

Clients MAY request the `OPTION_V4_ZEROTOUCH_REDIRECT` by including its option code in the Parameter Request List (55) in DHCP request messages.

On receipt of a DHCPv4 Reply message which contains the `OPTION_V4_ZEROTOUCH_REDIRECT`, the client performs the following steps:

1. Check the contents of the DHCPv4 message for at least one valid URI. If there is more than one valid URI in the list, a candidate list of possible URIs is created.
2. Attempt to connect to the one of the URIs in the candidate list. The order in which these are processed by the client is implementation specific and not defined here.

3. If a successful connection to the Zero Touch bootstrap server, then the client stops processing entries in the list and proceeds according to Section 6.3, step(3).
4. If the Zero Touch bootstrap server does not respond, provides an invalid response, or the transaction otherwise fails, the client SHOULD attempt to contact another server from the candidate list.

Any invalid URI entries received in the uri-data field are ignored by the client. If OPTION_V4_ZEROTOUCH_REDIRECT does not contain at least one valid URI entry in the uri-data field, then the client MUST discard the option.

DHCPv4 Server Behavior

The DHCPv4 server MAY include a single instance of Option OPTION_V4_ZEROTOUCH_REDIRECT in DHCP messages it sends. Servers MUST NOT send more than one instance of the OPTION_V4_ZEROTOUCH_REDIRECT option.

10.2. DHCPv6 Zero Touch Option

The DHCPv6 Zero Touch Option is used to provision the client with one or more URIs for bootstrap servers that can be contacted to attempt further configuration.

DHCPv6 Zero Touch Redirect Option

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|          option-code (TBD)          |          option-length          |
+-----+-----+-----+-----+-----+-----+-----+-----+
.          bootstrap-server-list (variable length)          .
+-----+-----+-----+-----+-----+-----+-----+-----+

```

- o option-code: OPTION_V6_ZEROTOUCH_REDIRECT (TBD)
- o option-length: The option length in octets
- o bootstrap-server-list: A list of servers for the client to attempt contacting, in order to obtain further bootstrapping data, in the format shown in [common-field-encoding].

DHCPv6 Client Behavior

Clients MAY request the OPTION_V6_ZEROTOUCH_REDIRECT option, as defined in [RFC3315], Sections 17.1.1, 18.1.1, 18.1.3, 18.1.4, 18.1.5, and 22.7. As a convenience to the reader, we mention here

that the client includes requested option codes in the Option Request Option.

On receipt of a DHCPv6 reply message which contains the `OPTION_V6_ZEROTOUCH_REDIRECT`, the client performs the following steps:

1. Check the contents of the DHCPv6 message for at least one valid URI. If there is more than one valid URI in the list, a candidate list of possible URIs is created.
2. Attempt to connect to the one of the URIs in the candidate list. The order in which these are processed by the client is implementation specific and not defined here.
3. If a successful connection to the Netconf Zero Touch Bootstrap server, then the client stops processing entries in the list and proceeds according to Section 6.3, step(3).
4. If the Zero Touch bootstrap server does not respond, provides and invalid response or the transaction otherwise fails, the client SHOULD attempt to contact another server from the candidate list.

Any invalid URI entries received in the uri-data field are ignored by the client. If `OPTION_V6_ZEROTOUCH_REDIRECT` does not contain at least one valid URI entry in the uri-data field, then the client MUST discard the option.

DHCPv6 Server Behavior

Sections 17.2.2 and 18.2 of [RFC3315] govern server operation in regard to option assignment. As a convenience to the reader, we mention here that the server will send a particular option code only if configured with specific values for that option code and if the client requested it.

Option `OPTION_V6_ZEROTOUCH_REDIRECT` is a singleton. Servers MUST NOT send more than one instance of the `OPTION_V6_ZEROTOUCH_REDIRECT` option.

10.3. Common Field Encoding

Both of the DHCPv4 and DHCPv6 options defined in this section encode a list of bootstrap server URIs. The 'URI' structure is an option that can contain multiple URIs (see [RFC7227], Section 5.7).

bootstrap-server-list:

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          uri-length          |          URI          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
```

- o uri-length: variable, in octets.
- o URI: URI of Netconf zerotouch bootstrap server, using the HTTPS URI scheme defined in Section 2.7.2 of RFC7230.

11. Security Considerations

11.1. Immutable storage for trust anchors

Devices MUST ensure that all their trust anchor certificates, including those for connecting to bootstrap servers and verifying ownership vouchers, are protected from external modification.

It may be necessary to update these certificates over time (e.g., the manufacturer wants to delegate trust to a new CA). It is therefore expected that devices MAY update these trust anchors when needed through a verifiable process, such as a software upgrade using signed software images.

11.2. Clock Sensitivity

The solution in this document relies on TLS certificates, owner certificates, and ownership vouchers, all of which require an accurate clock in order to be processed correctly (e.g., to test validity dates and revocation status). Implementations MUST ensure devices have an accurate clock when shipped from manufacturing facilities, and take steps to prevent clock tampering.

If it is not possible to ensure clock accuracy, it is RECOMMENDED that implementations disable the aspects of the solution having clock sensitivity. In particular, such implementations should assume that TLS certificates and owner certificates are not revokable. In real-world terms, this means that manufacturers SHOULD only issue a single ownership voucher for the lifetime of some devices.

It is important to note that implementations SHOULD NOT rely on NTP for time, as it is not a secure protocol.

11.3. Blindly authenticating a bootstrap server

This document allows a device to blindly authenticate a bootstrap server's TLS certificate. It does so to allow for cases where the redirect information may be obtained in an unsecured manner, which is desirable to support in some cases.

To compensate for this, this document requires that devices, when connected to an untrusted bootstrap server, do not send their IDevID certificate for client authentication, and they do not POST any progress notifications, and they assert that data downloaded from the server is signed.

11.4. Entropy loss over time

Section 7.2.7.2 of the IEEE Std 802.1AR-2009 standard says that IDevID certificate should never expire (i.e. having the notAfter value 99991231235959Z). Given the long-lived nature of these certificates, it is paramount to use a strong key length (e.g., 512-bit ECC).

11.5. Serial Numbers

This draft uses the device's serial number both in the IDevID certificate as well as in the bootstrap server API. Serial numbers are ubiquitous and prominently contained in invoices and on labels affixed to devices and their packaging. That said, serial numbers many times encode revealing information, such as the device's model number, manufacture date, and/or sequence number. Knowledge of this information may provide an adversary with details needed to launch an attack.

11.6. Sequencing Sources of Bootstrapping Data

For devices supporting more than one source for bootstrapping data, no particular sequencing order has to be observed for security reasons, as the solution for each source is considered equally secure. However, from a privacy perspective, it is RECOMMENDED that devices access local sources before accessing remote sources.

12. IANA Considerations

12.1. The BOOTP Manufacturer Extensions and DHCP Options Registry

IANA is kindly requested to allocate a new option code from the "BOOTP Manufacturer Extensions and DHCP Options" registry maintained at <http://www.iana.org/assignments/bootp-dhcp-parameters>:

TBD for OPTION_V4_ZEROTOUCH_REDIRECT

And a new option code from the "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)" registry maintained at <http://www.iana.org/assignments/dhcpv6-parameters>:

TBD for OPTION_V6_ZEROTOUCH_REDIRECT

12.2. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-zerotouch-information
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

12.3. The YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format defined in [RFC6020], the the following registrations are requested:

name: ietf-zerotouch-information
namespace: urn:ietf:params:xml:ns:yang:ietf-zerotouch-information
prefix: zt
reference: RFC XXXX

name: ietf-zerotouch-bootstrap-server
namespace: urn:ietf:params:xml:ns:yang:ietf-zerotouch-bootstrap-server
prefix: zt
reference: RFC XXXX

13. Other Considerations

Both this document and [I-D.ietf-anima-bootstrapping-keyinfra] define bootstrapping mechanisms. The authors have collaborated on both solutions and believe that each solution has merit and, in fact, can work together. That is, it is possible for a device to support both solutions simultaneously.

14. Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): David Harrington, Michael Behringer, Dean Bogdanovic, Martin Bjorklund, Joe Clarke, Toerless Eckert, Stephen Farrell, Stephen Hanna, Wes Hardaker, Russ Mundy, Reinaldo Penno, Randy Presuhn, Max Pritikin, Michael Richardson, Phil Shafer, Juergen Schoenwaelder.

Special thanks goes to Steve Hanna, Russ Mundy, and Wes Hardaker for brainstorming the original I-D's solution during the IETF 87 meeting in Berlin.

15. References

15.1. Normative References

- [I-D.ietf-anima-voucher]
Watsen, K., Richardson, M., Pritikin, M., and T. Eckert,
"Voucher Profile for Bootstrapping Protocols", draft-ietf-anima-voucher-03 (work in progress), June 2017.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<http://www.rfc-editor.org/info/rfc1035>>.
- [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>>.
- [RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998, <<http://www.rfc-editor.org/info/rfc2315>>.
- [RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <<http://www.rfc-editor.org/info/rfc3315>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<http://www.rfc-editor.org/info/rfc5280>>.

- [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>>.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <<http://www.rfc-editor.org/info/rfc6125>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<http://www.rfc-editor.org/info/rfc6234>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<http://www.rfc-editor.org/info/rfc6762>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<http://www.rfc-editor.org/info/rfc6763>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7227] Hankins, D., Mrugalski, T., Siodelski, M., Jiang, S., and S. Krishnan, "Guidelines for Creating New DHCPv6 Options", BCP 187, RFC 7227, DOI 10.17487/RFC7227, May 2014, <<http://www.rfc-editor.org/info/rfc7227>>.
- [RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <<http://www.rfc-editor.org/info/rfc7468>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.

[Std-802.1AR-2009]

IEEE SA-Standards Board, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.

15.2. Informative References

[I-D.ietf-anima-bootstrapping-keyinfra]

Pritikin, M., Richardson, M., Behringer, M., Bjarnason, S., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-06 (work in progress), May 2017.

[I-D.ietf-netconf-netconf-client-server]

Watsen, K., Wu, G., and J. Schoenwaelder, "NETCONF Client and Server Models", draft-ietf-netconf-netconf-client-server-03 (work in progress), June 2017.

[RFC2939] Droms, R., "Procedures and IANA Guidelines for Definition of New DHCP Options and Message Types", BCP 43, RFC 2939, DOI 10.17487/RFC2939, September 2000, <<http://www.rfc-editor.org/info/rfc2939>>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.

[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>>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <<http://www.rfc-editor.org/info/rfc6698>>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <<http://www.rfc-editor.org/info/rfc6960>>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<http://www.rfc-editor.org/info/rfc7317>>.

[RFC8071] Watsen, K., "NETCONF Call Home and RESTCONF Call Home",
RFC 8071, DOI 10.17487/RFC8071, February 2017,
<<http://www.rfc-editor.org/info/rfc8071>>.

Appendix A. Change Log

A.1. ID to 00

- o Major structural update; the essence is the same. Most every section was rewritten to some degree.
- o Added a Use Cases section
- o Added diagrams for "Actors and Roles" and "NMS Precondition" sections, and greatly improved the "Device Boot Sequence" diagram
- o Removed support for physical presence or any ability for configlets to not be signed.
- o Defined the Zero Touch Information DHCP option
- o Added an ability for devices to also download images from configuration servers
- o Added an ability for configlets to be encrypted
- o Now configuration servers only have to support HTTP/S - no other schemes possible

A.2. 00 to 01

- o Added boot-image and validate-owner annotations to the "Actors and Roles" diagram.
- o Fixed 2nd paragraph in section 7.1 to reflect current use of anyxml.
- o Added encrypted and signed-encrypted examples
- o Replaced YANG module with XSD schema
- o Added IANA request for the Zero Touch Information DHCP Option
- o Added IANA request for media types for boot-image and configuration

A.3. 01 to 02

- o Replaced the need for a configuration signer with the ability for each NMS to be able to sign its own configurations, using manufacturer signed ownership vouchers and owner certificates.

- o Renamed configuration server to bootstrap server, a more representative name given the information devices download from it.
- o Replaced the concept of a configlet by defining a southbound interface for the bootstrap server using YANG.
- o Removed the IANA request for the boot-image and configuration media types

A.4. 02 to 03

- o Minor update, mostly just to add an Editor's Note to show how this draft might integrate with the draft-pritikin-anima-bootstrapping-keyinfra.

A.5. 03 to 04

- o Major update formally introducing unsigned data and support for Internet-based redirect servers.
- o Added many terms to Terminology section.
- o Added all new "Guiding Principles" section.
- o Added all new "Sources for Bootstrapping Data" section.
- o Rewrote the "Interactions" section and renamed it "Workflow Overview".

A.6. 04 to 05

- o Semi-major update, refactoring the document into more logical parts
- o Created new section for information types
- o Added support for DNS servers
- o Now allows provisional TLS connections
- o Bootstrapping data now supports scripts
- o Device Details section overhauled
- o Security Considerations expanded
- o Filled in enumerations for notification types

A.7. 05 to 06

- o Minor update
- o Added many Normative and Informative references.
- o Added new section Other Considerations.

A.8. 06 to 07

- o Minor update
- o Added an Editorial Note section for RFC Editor.
- o Updated the IANA Considerations section.

A.9. 07 to 08

- o Minor update
- o Updated to reflect review from Michael Richardson.

A.10. 08 to 09

- o Added in missing "Signature" artifact example.
- o Added recommendation for manufacturers to use interoperable formats and file naming conventions for removable storage devices.
- o Added configuration-handling leaf to guide if config should be merged, replaced, or processed like an edit-config/yang-patch document.
- o Added a pre-configuration script, in addition to the post-configuration script from -05 (issue #15).

A.11. 09 to 10

- o Factored ownership voucher and voucher revocation to a separate document: draft-kwatsen-netconf-voucher. (issue #11)
- o Removed <configuration-handling> options 'edit-config' and yang-patch'. (issue #12)
- o Defined how a signature over signed-data returned from a bootstrap server is processed. (issue #13)

- o Added recommendation for removable storage devices to use open/standard file systems when possible. (issue #14)
- o Replaced notifications "script-[warning/error]" with "[pre/post]-script-[warning/error]". (goes with issue #15)
- o switched owner-certificate to be encoded using the pkcs#7 format. (issue #16)
- o Replaced md5/sha1 with sha256 inside a choice statement, for future extensibility. (issue #17)
- o A ton of editorial changes, as I went thru the entire draft with a fine-toothed comb.

A.12. 10 to 11

- o fixed yang validation issues found by IETFYANGPageCompilation. note: these issues were NOT found by pyang --ietf or by the submission-time validator...
- o fixed a typo in the yang module, someone the config false statement was removed.

A.13. 11 to 12

- o fixed typo that prevented Appendix B from loading the examples correctly.
- o fixed more yang validation issues found by IETFYANGPageCompilation. note: again, these issues were NOT found by pyang --ietf or by the submission-time validator...
- o updated a few of the notification enumerations to be more consistent with the other enumerations (following the warning/error pattern).
- o updated the information-type artifact to state how it's encoded, matching the language that was in Appendix B.

A.14. 12 to 13

- o defined a standalone artifact to encode the old information-type into a PKCS#7 structure.
- o standalone information artifact hardcodes JSON encoding (to match the voucher draft).

- o combined the information and signature PKCS#7 structures into a single PKCS#7 structure.
- o moved the certificate-revocations into the owner-certificate's PKCS#7 structure.
- o eliminated support for voucher-revocations, to reflect the voucher-draft's switch from revocations to renewals.

A.15. 13 to 14

- o Renamed "bootstrap information" to "onboarding information".
- o Rewrote DHCP sections to address the packet-size limitation issue, as discussed in Chicago.
- o Added Ian as an author for his text-contributions to the DHCP sections.
- o Removed the Guiding Principles section.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Mikael Abrahamsson
T-Systems

EMail: mikael.abrahamsson@t-systems.se

Ian Farrer
Deutsche Telekom AG

EMail: ian.farrer@telekom.de

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

M. Bjorklund
Tail-f Systems
J. Schoenwaelder
Jacobs University
P. Shafer
K. Watsen
Juniper Networks
R. Wilton
Cisco Systems
July 3, 2017

Network Management Datastore Architecture
draft-ietf-netmod-revised-datastores-03

Abstract

Datastores are a fundamental concept binding the data models written in the YANG data modeling language to network management protocols such as NETCONF and RESTCONF. This document defines an architectural framework for datastores based on the experience gained with the initial simpler model, addressing requirements that were not well supported in the initial model.

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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 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	3
2. Terminology	3
3. Background	5
3.1. Original Model of Datastores	6
4. Architectural Model of Datastores	8
4.1. The Startup Configuration Datastore (<startup>)	9
4.2. The Candidate Configuration Datastore (<candidate>)	10
4.3. The Running Configuration Datastore (<running>)	10
4.4. The Intended Configuration Datastore (<intended>)	10
4.5. Conventional Configuration Datastores	11
4.6. Dynamic Datastores	11
4.7. The Operational State Datastore (<operational>)	11
4.7.1. Missing Resources	12
4.7.2. System-controlled Resources	13
4.7.3. Origin Metadata Annotation	13
5. Implications on YANG	14
5.1. XPath Context	14
6. YANG Modules	15
7. IANA Considerations	21
7.1. Updates to the IETF XML Registry	21
7.2. Updates to the YANG Module Names Registry	22
8. Security Considerations	22
9. Acknowledgments	22
10. References	23
10.1. Normative References	23
10.2. Informative References	23
Appendix A. Guidelines for Defining Datastores	24
A.1. Define which YANG modules can be used in the datastore	24
A.2. Define which subset of YANG-modeled data applies	25
A.3. Define how data is actualized	25
A.4. Define which protocols can be used	25
A.5. Define YANG identities for the datastore	25
Appendix B. Ephemeral Dynamic Datastore Example	25
Appendix C. Example Data	27
C.1. System Example	27
C.2. BGP Example	29
C.2.1. Datastores	31
C.2.2. Adding a Peer	31

C.2.3. Removing a Peer	32
C.3. Interface Example	33
C.3.1. Pre-provisioned Interfaces	33
C.3.2. System-provided Interface	34
Authors' Addresses	35

1. Introduction

This document provides an architectural framework for datastores as they are used by network management protocols such as NETCONF [RFC6241], RESTCONF [RFC8040] and the YANG [RFC7950] data modeling language. Datastores are a fundamental concept binding network management data models to network management protocols. Agreement on a common architectural model of datastores ensures that data models can be written in a network management protocol agnostic way. This architectural framework identifies a set of conceptual datastores but it does not mandate that all network management protocols expose all these conceptual datastores. This architecture is agnostic with regard to the encoding used by network management protocols.

2. Terminology

This document defines the following terms:

- o datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof. A datastore maps to an instantiated YANG data tree.
- o configuration: Data that is required to get a device from its initial default state into a desired operational state. This data is modeled in YANG using "config true" nodes. Configuration can originate from different sources.
- o configuration datastore: A datastore holding configuration.
- o running configuration datastore: A configuration datastore holding the current configuration of the device. It may include inactive configuration or template-mechanism-oriented configuration that require further expansion. This datastore is commonly referred to as "<running>".
- o candidate configuration datastore: A configuration datastore that can be manipulated without impacting the device's running configuration datastore and that can be committed to the running configuration datastore. This datastore is commonly referred to as "<candidate>".

- o startup configuration datastore: A configuration datastore holding the configuration loaded by the device into the running configuration datastore when it boots. This datastore is commonly referred to as "<startup>".
- o intended configuration: Configuration that is intended to be used by the device. For example, intended configuration excludes any inactive configuration and it would include configuration produced through the expansion of templates.
- o intended configuration datastore: A configuration datastore holding the complete intended configuration of the device. This datastore is commonly referred to as "<intended>".
- o conventional configuration datastore: One of the following set of configuration datastores: <running>, <startup>, <candidate>, and <intended>. These datastores share a common schema and protocol operations allow copying data between these datastores. The term "conventional" is chosen as a generic umbrella term for these datastores.
- o conventional configuration: Configuration that is stored in any of the conventional configuration datastores.
- o dynamic datastore: A datastore holding data obtained dynamically during the operation of a device through interaction with other systems, rather than through one of the conventional configuration datastores.
- o dynamic configuration: Configuration obtained via a dynamic datastore.
- o learned configuration: Configuration that has been learned via protocol interactions with other systems that is not conventional or dynamic configuration.
- o system configuration: Configuration that is supplied by the device itself.
- o default configuration: Configuration that is not explicitly provided but for which a value defined in the data model is used.
- o applied configuration: Configuration that is actively in use by a device. Applied configuration originates from conventional, dynamic, learned, system and default configuration.
- o system state: The additional data on a system that is not configuration, such as read-only status information and collected

statistics. System state is transient and modified by interactions with internal components or other systems. System state is modeled in YANG using "config false" nodes.

- o operational state: The combination of applied configuration and system state.
- o operational state datastore: A datastore holding the complete operational state of the device. This datastore is commonly referred to as "<operational>".
- o origin: A metadata annotation indicating the origin of a data item.
- o remnant configuration: Configuration that remains part of the applied configuration for a period of time after it has been removed from the intended configuration or dynamic configuration. The time period may be minimal, or may last until all resources used by the newly-deleted configuration (e.g., network connections, memory allocations, file handles) have been deallocated.

The following additional terms are not datastore specific but commonly used and thus defined here as well:

- o client: An entity that can access YANG-defined data on a server, over some network management protocol.
- o server: An entity that provides access to YANG-defined data to a client, over some network management protocol.
- o notification: A server-initiated message indicating that a certain event has been recognized by the server.
- o remote procedure call: An operation that can be invoked by a client on a server.

3. Background

NETCONF [RFC6241] provides the following definitions:

- o datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof.
- o configuration datastore: The datastore holding the complete set of configuration that is required to get a device from its initial default state into a desired operational state.

YANG 1.1 [RFC7950] provides the following refinements when NETCONF is used with YANG (which is the usual case but note that NETCONF was defined before YANG existed):

- o datastore: When modeled with YANG, a datastore is realized as an instantiated data tree.
- o configuration datastore: When modeled with YANG, a configuration datastore is realized as an instantiated data tree with configuration.

[RFC6244] defined operational state data as follows:

- o Operational state data is a set of data that has been obtained by the system at runtime and influences the system's behavior similar to configuration data. In contrast to configuration data, operational state is transient and modified by interactions with internal components or other systems via specialized protocols.

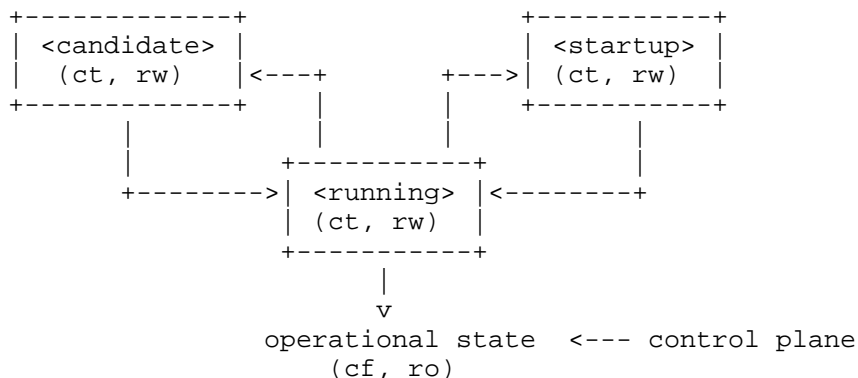
Section 4.3.3 of [RFC6244] discusses operational state and among other things mentions the option to consider operational state as being stored in another datastore. Section 4.4 of this document then concludes that at the time of the writing, modeling state as distinct leafs and distinct branches is the recommended approach.

Implementation experience and requests from operators [I-D.ietf-netmod-opstate-reqs], [I-D.openconfig-netmod-opstate] indicate that the datastore model initially designed for NETCONF and refined by YANG needs to be extended. In particular, the notion of intended configuration and applied configuration has developed.

Furthermore, separating operational state from configuration in a separate branch in the data model has been found operationally complicated, and typically impacts the readability of module definitions due to overuse of groupings. The relationship between the branches is not machine readable and filter expressions operating on configuration and on related operational state are different.

3.1. Original Model of Datastores

The following drawing shows the original model of datastores as it is currently used by NETCONF [RFC6241]:



ct = config true; cf = config false
 rw = read-write; ro = read-only
 boxes denote datastores

Note that this diagram simplifies the model: read-only (ro) and read-write (rw) is to be understood at a conceptual level. In NETCONF, for example, support for <candidate> and <startup> is optional and <running> does not have to be writable. Furthermore, <startup> can only be modified by copying <running> to <startup> in the standardized NETCONF datastore editing model. The RESTCONF protocol does not expose these differences and instead provides only a writable unified datastore, which hides whether edits are done through <candidate> or by directly modifying <running> or via some other implementation specific mechanism. RESTCONF also hides how configuration is made persistent. Note that implementations may also have additional datastores that can propagate changes to <running>. NETCONF explicitly mentions so called named datastores.

Some observations:

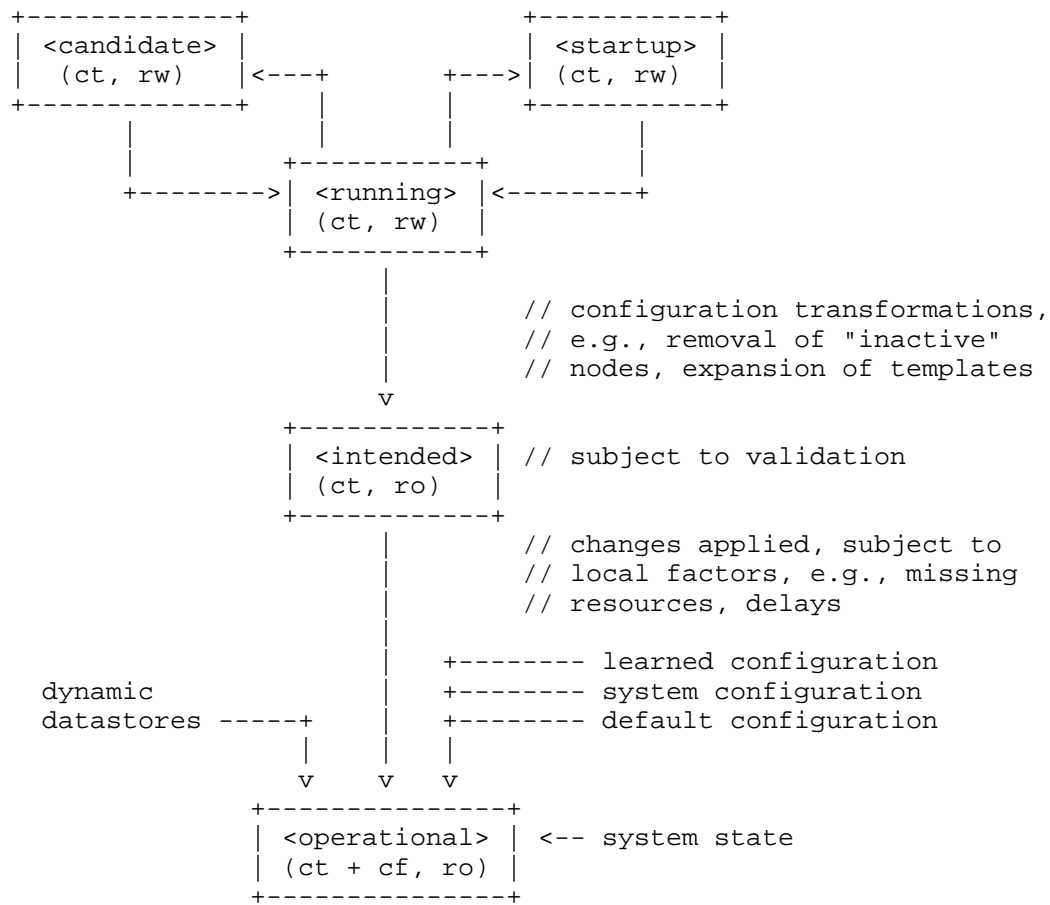
- o Operational state has not been defined as a datastore although there were proposals in the past to introduce an operational state datastore.
- o The NETCONF <get/> operation returns the content of the <running> configuration datastore together with the operational state. It is therefore necessary that "config false" data is in a different branch than the "config true" data if the operational state can have a different lifetime compared to configuration or if configuration is not immediately or successfully applied.
- o Several implementations have proprietary mechanisms that allow clients to store inactive data in <running>; this inactive data is only exposed to clients that indicate that they support the

concept of inactive data; clients not indicating support for inactive data receive the content of <running> with the inactive data removed. Inactive data is conceptually removed before validation.

- o Some implementations have proprietary mechanisms that allow clients to define configuration templates in <running>. These templates are expanded automatically by the system, and the resulting configuration is applied internally.
- o Some operators have reported that it is essential for them to be able to retrieve the configuration that has actually been successfully applied, which may be a subset or a superset of the <running> configuration.

4. Architectural Model of Datastores

Below is a new conceptual model of datastores extending the original model in order to reflect the experience gained with the original model.



```
ct = config true; cf = config false
rw = read-write; ro = read-only
boxes denote named datastores
```

4.1. The Startup Configuration Datastore (<startup>)

The startup configuration datastore (<startup>) is an optional configuration datastore holding the configuration loaded by the device when it boots. <startup> is only present on devices that separate the startup configuration from the running configuration datastore.

The startup configuration datastore may not be supported by all protocols or implementations.

On devices that support non-volatile storage, the contents of `<startup>` will typically persist across reboots via that storage. At boot time, the device loads the saved startup configuration into `<running>`. To save a new startup configuration, data is copied to `<startup>`, either via implicit or explicit protocol operations.

4.2. The Candidate Configuration Datastore (`<candidate>`)

The candidate configuration datastore (`<candidate>`) is an optional configuration datastore that can be manipulated without impacting the device's current configuration and that can be committed to `<running>`.

The candidate configuration datastore may not be supported by all protocols or implementations.

`<candidate>` does not typically persist across reboots, even in the presence of non-volatile storage. If `<candidate>` is stored using non-volatile storage, it should be reset at boot time to the contents of `<running>`.

4.3. The Running Configuration Datastore (`<running>`)

The running configuration datastore (`<running>`) holds the complete current configuration on the device. It may include inactive configuration or template-mechanism-oriented configuration that require further expansion.

If a device does not have a distinct `<startup>` and non-volatile is available, the device will typically use that non-volatile storage to allow `<running>` to persist across reboots.

4.4. The Intended Configuration Datastore (`<intended>`)

The intended configuration datastore (`<intended>`) is a read-only configuration datastore. It is tightly coupled to `<running>`. When data is written to `<running>`, the data that is to be validated is also conceptually written to `<intended>`. Validation is performed on the contents of `<intended>`.

For simple implementations, `<running>` and `<intended>` are identical.

`<intended>` does not persist across reboots; its relationship with `<running>` makes that unnecessary.

Currently there are no standard mechanisms defined that affect `<intended>` so that it would have different contents than `<running>`, but this architecture allows for such mechanisms to be defined.

One example of such a mechanism is support for marking nodes as inactive in <running>. Inactive nodes are not copied to <intended>, and are thus not taken into account when validating the configuration.

Another example is support for templates. Templates are expanded when copied into <intended>, and the expanded result is validated.

4.5. Conventional Configuration Datastores

The conventional configuration datastores are a set of configuration datastores that share a common schema, allowing data to be copied between them. The term is meant as a generic umbrella description of these datastores. The set of datastores include:

- o <running>
- o <candidate>
- o <startup>
- o <intended>

Other conventional configuration datastores may be defined in future documents.

The flow of data between these datastores is depicted in section Section 4.

The specific protocols may define explicit operations to copy between these datastores, e.g., NETCONF's <copy-config> operation.

4.6. Dynamic Datastores

The model recognizes the need for dynamic datastores that are, by definition, not part of the persistent configuration of a device. In some contexts, these have been termed ephemeral datastores since the information is ephemeral, i.e., lost upon reboot. The dynamic datastores interact with the rest of the system through <operational>.

4.7. The Operational State Datastore (<operational>)

The operational state datastore (<operational>) is a read-only datastore that consists of all "config true" and "config false" nodes defined in the schema. In the original NETCONF model the operational state only had "config false" nodes. The reason for incorporating

"config true" nodes here is to be able to expose all operational settings without having to replicate definitions in the data models.

<operational> contains system state and all configuration actually used by the system. This includes all applied configuration from <intended>, system-provided configuration, and default values defined by any supported data models. In addition, <operational> also contains applied data from dynamic datastores.

Requests to retrieve nodes from <operational> always return the value in use if the node exists, regardless of any default value specified in the YANG module. If no value is returned for a given node, then this implies that the node is not used by the device.

<operational> does not persist across reboots.

Changes to configuration may take time to percolate through to <operational>. During this period, <operational> may contain nodes for both the previous and current configuration, as closely as possible tracking the current operation of the device. Such remnant configuration from the previous configuration persists until the system has released resources used by the newly-deleted configuration (e.g., network connections, memory allocations, file handles).

As a result of remnant configuration, the semantic constraints defined in the data model cannot be relied upon for <operational>, since the system may have remnant configuration whose constraints were valid with the previous configuration and that are not valid with the current configuration. Since constraints on "config false" nodes may refer to "config true" nodes, remnant configuration may force the violation of those constraints. The constraints that may not hold include "when", "must", "min-elements", and "max-elements". Note that syntactic constraints cannot be violated, including hierarchical organization, identifiers, and type-based constraints.

4.7.1. Missing Resources

Configuration in <intended> can refer to resources that are not available or otherwise not physically present. In these situations, these parts of the <intended> configuration are not applied. The data appears in <intended> but does not appear in <operational>.

A typical example is an interface configuration that refers to an interface that is not currently present. In such a situation, the interface configuration remains in <intended> but the interface configuration will not appear in <operational>.

Note that configuration validity cannot depend on the current state of such resources, since that would imply the removing a resource might render the configuration invalid. This is unacceptable, especially given that rebooting such a device would fail to boot due to an invalid configuration. Instead we allow configuration for missing resources to exist in <running> and <intended>, but it will not appear in <operational>.

4.7.2. System-controlled Resources

Sometimes resources are controlled by the device and the corresponding system controlled data appear in (and disappear from) <operational> dynamically. If a system controlled resource has matching configuration in <intended> when it appears, the system will try to apply the configuration, which causes the configuration to appear in <operational> eventually (if application of the configuration was successful).

4.7.3. Origin Metadata Annotation

As data flows into <operational>, it is conceptually marked with a metadata annotation ([RFC7952]) that indicates its origin. The origin applies to all data nodes except non-presence containers. The "origin" metadata annotation is defined in Section 6. The values are YANG identities. The following identities are defined:

- o origin: abstract base identity from which the other origin identities are derived.
- o intended: represents data provided by <intended>.
- o dynamic: represents data provided by a dynamic datastore.
- o system: represents data provided by the system itself, including both system configuration and system state. Examples of system configuration include applied configuration for an always existing loopback interface, or interface configuration that is auto-created due to the hardware currently present in the device.
- o learned: represents configuration that has been learned via protocol interactions with other systems, including protocols such as link-layer negotiations, routing protocols, DHCP, etc.
- o default: represents data using a default value specified in the data model, using either values in the "default" statement or any values described in the "description" statement. The default origin is only used when the data has not been provided by any other source.

- o unknown: represents data for which the system cannot identify the origin.

These identities can be further refined, e.g., there could be separate identities for particular types or instances of dynamic datastore derived from "dynamic".

In all cases, the device should report the origin that most accurately reflects the source of the data that is actively being used by the system.

In cases where it could be ambiguous as to which origin should be used, i.e. where the same data node value has originated from multiple sources, then the description statement in the YANG module should be used as guidance for choosing the appropriate origin. For example:

If for a particular configuration node, the associated YANG description statement indicates that a protocol negotiated value overrides any configured value, then the origin would be reported as "learned", even when a learned value is the same as the configured value.

Conversely, if for a particular configuration node, the associated YANG description statement indicates that a protocol negotiated value does not override an explicitly configured value, then the origin would be reported as "intended" even when a learned value is the same as the configured value.

In the case that a device cannot provide an accurate origin for a particular data node then it should use the origin "unknown".

5. Implications on YANG

5.1. XPath Context

If a server implements the architecture defined in this document, the accessible trees for some XPath contexts are refined as follows:

- o If the XPath expression is defined in a substatement to a data node that represents system state, the accessible tree is all operational state in the server. The root node has all top-level data nodes in all modules as children.
- o If the XPath expression is defined in a substatement to a "notification" statement, the accessible tree is the notification instance and all operational state in the server. If the notification is defined on the top level in a module, then the

root node has the node representing the notification being defined and all top-level data nodes in all modules as children. Otherwise, the root node has all top-level data nodes in all modules as children.

- o If the XPath expression is defined in a substatement to an "input" statement in an "rpc" or "action" statement, the accessible tree is the RPC or action operation instance and all operational state in the server. The root node has top-level data nodes in all modules as children. Additionally, for an RPC, the root node also has the node representing the RPC operation being defined as a child. The node representing the operation being defined has the operation's input parameters as children.
- o If the XPath expression is defined in a substatement to an "output" statement in an "rpc" or "action" statement, the accessible tree is the RPC or action operation instance and all operational state in the server. The root node has top-level data nodes in all modules as children. Additionally, for an RPC, the root node also has the node representing the RPC operation being defined as a child. The node representing the operation being defined has the operation's output parameters as children.

6. YANG Modules

<CODE BEGINS> file "ietf-datastores@2017-04-26.yang"

```
module ietf-datastores {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-datastores";
  prefix ds;

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

  contact
    "WG Web:    <https://datatracker.ietf.org/wg/netmod/>

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

    Author:     Martin Bjorklund
                <mailto:mbj@tail-f.com>

    Author:     Juergen Schoenwaelder
                <mailto:j.schoenwaelder@jacobs-university.de>

    Author:     Phil Shafer
                <mailto:phil@juniper.net>
```

Author: Kent Watsen
<mailto:kwatsen@juniper.net>

Author: Rob Wilton
<rwilton@cisco.com>";

description

"This YANG module defines two sets of identities for datastores. The first identifies the datastores themselves, the second identifies are for datastore propterties.

Copyright (c) 2017 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<http://www.rfc-editor.org/info/rfcxxxx>); see the RFC itself for full legal notices.";

```
revision 2017-04-26 {  
  description  
    "Initial revision.";  
  reference  
    "RFC XXXX: Network Management Datastore Architecture";  
}
```

```
/*  
 * Identities  
 */
```

```
identity datastore {  
  description  
    "Abstract base identity for datastore identities.";  
}
```

```
identity conventional {  
  base datastore;  
  description  
    "Abstract base identity for conventional configuration  
    datastores.";  
}
```

```
identity running {
  base conventional;
  description
    "The running configuration datastore.";
}

identity candidate {
  base conventional;
  description
    "The candidate configuration datastore.";
}

identity startup {
  base conventional;
  description
    "The startup configuration datastore.";
}

identity intended {
  base conventional;
  description
    "The intended configuration datastore.";
}

identity dynamic {
  base datastore;
  description
    "Abstract base identity for dynamic datastores.";
}

identity operational {
  base datastore;
  description
    "The operational state datastore.";
}

identity property {
  description
    "Abstract base identity for datastore identities.";
}

identity writable {
  base property;
  description
    "Used on the 'running' datastore to indicate that it can be
    written to.";
```

```
}  
  
identity auto-persist {  
  base property;  
  description  
    "Used on the 'running' datastore to indicate that writes to  
    it will be automatically persisted.  
  
    If the 'startup' datastore is also supported, clients may  
    query its contents to ensure its synchronization.  
  
    If the 'startup' datastore is not supported, and this  
    property is not set, then clients must use a mechanism  
    provided by the protocol to explicitly persist the  
    'running' datastore's contents.";  
}  
  
identity rollback-on-error {  
  base property;  
  description  
    "Used on either the 'running' or 'candidate' datastores to  
    indicate that clients may request atomic update behavior.";  
}  
  
identity confirmed-commit {  
  base property;  
  description  
    "Used on the 'candidate' datastore to indicate that  
    clients may request confirmed-commit update behavior.";  
}  
  
identity validate {  
  base property;  
  description  
    "Used on the 'candidate' datastore to indicate that  
    clients may request datastore validation.";  
}  
}
```

<CODE ENDS>

<CODE BEGINS> file "ietf-origin@2017-04-26.yang"

```
module ietf-origin {  
  yang-version 1.1;  
  namespace "urn:ietf:params:xml:ns:yang:ietf-origin";
```

```
prefix or;

import ietf-yang-metadata {
  prefix md;
}

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

contact
  "WG Web:    <https://datatracker.ietf.org/wg/netmod/>

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

  Author:     Martin Bjorklund
              <mailto:mbj@tail-f.com>

  Author:     Juergen Schoenwaelder
              <mailto:j.schoenwaelder@jacobs-university.de>

  Author:     Phil Shafer
              <mailto:phil@juniper.net>

  Author:     Kent Watsen
              <mailto:kwatsen@juniper.net>

  Author:     Rob Wilton
              <rwilton@cisco.com>";

description
  "This YANG module defines an 'origin' metadata annotation, and a
  set of identities for the origin value.

  Copyright (c) 2017 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject to
  the license terms contained in, the Simplified BSD License set
  forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX
  (http://www.rfc-editor.org/info/rfcxxxx); see the RFC itself
  for full legal notices.";

revision 2017-04-26 {
```

```
    description
        "Initial revision.";
    reference
        "RFC XXXX: Network Management Datastore Architecture";
}

/*
 * Identities
 */

identity origin {
    description
        "Abstract base identity for the origin annotation.";
}

identity intended {
    base origin;
    description
        "Denotes data from the intended configuration datastore";
}

identity dynamic {
    base origin;
    description
        "Denotes data from a dynamic datastore.";
}

identity system {
    base origin;
    description
        "Denotes data originated by the system itself, including
        both system configuration and system state.

        Examples of system configuration include applied configuration
        for an always existing loopback interface, or interface
        configuration that is auto-created due to the hardware
        currently present in the device.";
}

identity learned {
    base origin;
    description
        "Denotes configuration learned from protocol interactions with
        other devices, instead of via the intended configuration
        datastore or any dynamic datastore.

        Examples of protocols that provide learned configuration
        include link-layer negotiations, routing protocols, and
```



```
        DHCP.";
    }

    identity default {
        base origin;
        description
            "Denotes data that does not have an configured or learned
            value, but has a default value in use.  Covers both values
            defined in a 'default' statement, and values defined via an
            explanation in a 'description' statement.";
    }

    identity unknown {
        base origin;
        description
            "Denotes data for which the system cannot identify the
            origin.";
    }

    /*
     * Metadata annotations
     */

    md:annotation origin {
        type identityref {
            base origin;
        }
        description
            "The 'origin' annotation can be present on any node in a
            datastore.  It specifies from where the node originated.";
    }
}

<CODE ENDS>
```

7. IANA Considerations

7.1. Updates to the IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-datastores
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-origin
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

7.2. Updates to the YANG Module Names Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the the following registrations are requested:

name:	ietf-datastores
namespace:	urn:ietf:params:xml:ns:yang:ietf-datastores
prefix:	ds
reference:	RFC XXXX
name:	ietf-origin
namespace:	urn:ietf:params:xml:ns:yang:ietf-origin
prefix:	or
reference:	RFC XXXX

8. Security Considerations

This document discusses an architectural model of datastores for network management using NETCONF/RESTCONF and YANG. It has no security impact on the Internet.

9. Acknowledgments

This document grew out of many discussions that took place since 2010. Several Internet-Drafts ([I-D.bjorklund-netmod-operational], [I-D.wilton-netmod-opstate-yang], [I-D.ietf-netmod-opstate-reqs], [I-D.kwatsen-netmod-opstate], [I-D.openconfig-netmod-opstate]) and [RFC6244] touched on some of the problems of the original datastore model. The following people were authors to these Internet-Drafts or otherwise actively involved in the discussions that led to this document:

- o Lou Berger, LabN Consulting, L.L.C., <lberger@labn.net>
- o Andy Bierman, YumaWorks, <andy@yumaworks.com>
- o Marcus Hines, Google, <hines@google.com>
- o Christian Hopps, Deutsche Telekom, <chopps@chopps.org>

- o Acee Lindem, Cisco Systems, <acee@cisco.com>
- o Ladislav Lhotka, CZ.NIC, <lhotka@nic.cz>
- o Thomas Nadeau, Brocade Networks, <tnadeau@lucidvision.com>
- o Anees Shaikh, Google, <aashaikh@google.com>
- o Rob Shakir, Google, <robjs@google.com>

Juergen Schoenwaelder was partly funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its Seventh Framework Programme.

10. References

10.1. Normative References

- [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>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <<http://www.rfc-editor.org/info/rfc7952>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

10.2. Informative References

- [I-D.bjorklund-netmod-operational] Bjorklund, M. and L. Lhotka, "Operational Data in NETCONF and YANG", draft-bjorklund-netmod-operational-00 (work in progress), October 2012.
- [I-D.ietf-netmod-opstate-reqs] Watsen, K. and T. Nadeau, "Terminology and Requirements for Enhanced Handling of Operational State", draft-ietf-netmod-opstate-reqs-04 (work in progress), January 2016.

- [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-02 (work in progress), February 2016.
- [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.
- [I-D.wilton-netmod-opstate-yang]
Wilton, R., "With-config-state" Capability for NETCONF/RESTCONF", draft-wilton-netmod-opstate-yang-02 (work in progress), December 2015.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [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>>.
- [RFC6244] Shafer, P., "An Architecture for Network Management Using NETCONF and YANG", RFC 6244, DOI 10.17487/RFC6244, June 2011, <<http://www.rfc-editor.org/info/rfc6244>>.

Appendix A. Guidelines for Defining Datastores

The definition of a new datastore in this architecture should be provided in a document (e.g., an RFC) purposed to the definition of the datastore. When it makes sense, more than one datastore may be defined in the same document (e.g., when the datastores are logically connected). Each datastore's definition should address the points specified in the sections below.

A.1. Define which YANG modules can be used in the datastore

Not all YANG modules may be used in all datastores. Some datastores may constrain which data models can be used in them. If it is desirable that a subset of all modules can be targeted to the datastore, then the documentation defining the datastore must indicate this.

A.2. Define which subset of YANG-modeled data applies

By default, the data in a datastore is modeled by all YANG statements in the available YANG modules. However, it is possible to specify criteria that YANG statements must satisfy in order to be present in a datastore. For instance, maybe only "config true" nodes are present, or "config false nodes" that also have a specific YANG extension (e.g., "i2rs:ephemeral true") are present in the datastore.

A.3. Define how data is actualized

The new datastore must specify how it interacts with other datastores. For example, the diagram in Section 4 depicts dynamic datastores feeding into <operational>. How this interaction occurs must be defined by any dynamic datastore. In some cases, it may occur implicitly, as soon as the data is put into the dynamic datastore while, in other cases, an explicit action (e.g., an RPC) may be required to trigger the application of the datastore's data.

A.4. Define which protocols can be used

By default, it is assumed that both the NETCONF and RESTCONF protocols can be used to interact with a datastore. However, it may be that only a specific protocol can be used (e.g., ForCES) or that a subset of all protocol operations or capabilities are available (e.g., no locking or no XPath-based filtering).

A.5. Define YANG identities for the datastore

The datastore must be defined with a YANG identity that uses the "ds:datastore" identity or one of its derived identities as its base. This identity is necessary so that the datastore can be referenced in protocol operations (e.g., <get-data>).

The datastore may also be defined with an identity that uses the "or:origin" identity or one its derived identities as its base. This identity is needed if the datastore interacts with <operational> so that data originating from the datastore can be identified as such via the "origin" metadata attribute defined in Section 6.

An example of these guidelines in use is provided in Appendix B.

Appendix B. Ephemeral Dynamic Datastore Example

The section defines documentation for an example dynamic datastore using the guidelines provided in Appendix A. While this example is very terse, it is expected to be that a standalone RFC would be needed when fully expanded.

This example defines a dynamic datastore called "ephemeral", which is loosely modeled after the work done in the I2RS working group.

1. Name : ephemeral
2. YANG modules : all (default)
3. YANG statements : config false + ephemeral true
4. How applied : automatic
5. Protocols : NC/RC (default)
6. YANG Module : (see below)

```
module example-ds-ephemeral {
  yang-version 1.1;
  namespace "urn:example:ds-ephemeral";
  prefix eph;

  import ietf-datastores {
    prefix ds;
  }
  import ietf-origin {
    prefix or;
  }

  // add datastore identity
  identity ds-ephemeral {
    base ds:datastore;
    description
      "The 'ephemeral' datastore.";
  }

  // add origin identity
  identity or-ephemeral {
    base or:dynamic;
    description
      "Denotes data from the ephemeral dynamic datastore.";
  }

  // define ephemeral extension
  extension ephemeral {
    argument "value";
    description
      "This extension is mixed into config false YANG nodes to
      indicate that they are writable nodes in the 'ephemeral'
      datastore. This statement takes a single argument
      representing a boolean having the values 'true' and
      'false'. The default value is 'false'.";
  }
}
```

Appendix C. Example Data

The use of datastores is complex, and many of the subtle effects are more easily presented using examples. This section presents a series of example data models with some sample contents of the various datastores.

C.1. System Example

In this example, the following fictional module is used:

```
module example-system {
  yang-version 1.1;
  namespace urn:example:system;
  prefix sys;

  import ietf-inet-types {
    prefix inet;
  }

  container system {
    leaf hostname {
      type string;
    }

    list interface {
      key name;

      leaf name {
        type string;
      }

      container auto-negotiation {
        leaf enabled {
          type boolean;
          default true;
        }
        leaf speed {
          type uint32;
          units mbps;
          description
            "The advertised speed, in mbps.";
        }
      }
    }

    leaf speed {
      type uint32;
      units mbps;
    }
  }
}
```

```

        config false;
        description
            "The speed of the interface, in mbps.";
    }

    list address {
        key ip;

        leaf ip {
            type inet:ip-address;
        }
        leaf prefix-length {
            type uint8;
        }
    }
}
}
}

```

The operator has configured the host name and two interfaces, so the contents of <intended> is:

```

<system xmlns="urn:example:system">

  <hostname>foo</hostname>

  <interface>
    <name>eth0</name>
    <auto-negotiation>
      <speed>1000</speed>
    </auto-negotiation>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>32</prefix-length>
    </address>
  </interface>

  <interface>
    <name>eth1</name>
    <address>
      <ip>2001:db8::20</ip>
      <prefix-length>32</prefix-length>
    </address>
  </interface>

</system>

```


The system has detected that the hardware for one of the configured interfaces ("eth1") is not yet present, so the configuration for that interface is not applied. Further, the system has received a host name and an additional IP address for "eth0" over DHCP. In addition to a default value, a loopback interface is automatically added by the system, and the result of the "speed" auto-negotiation. All of this is reflected in <operational>:

```
<system
  xmlns="urn:example:system"
  xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">

  <hostname or:origin="or:dynamic">bar</hostname>

  <interface or:origin="or:intended">
    <name>eth0</name>
    <auto-negotiation>
      <enabled or:origin="or:default">true</enabled>
      <speed>1000</speed>
    </auto-negotiation>
    <speed>100</speed>
    <address>
      <ip>2001:db8::10</ip>
      <prefix-length>64</prefix-length>
    </address>
    <address or:origin="or:dynamic">
      <ip>2001:db8::1:100</ip>
      <prefix-length>64</prefix-length>
    </address>
  </interface>

  <interface or:origin="or:system">
    <name>lo0</name>
    <address>
      <ip>::1</ip>
      <prefix-length>128</prefix-length>
    </address>
  </interface>

</system>
```

C.2. BGP Example

Consider the following piece of a ersatz BGP module:

```

container bgp {
  leaf local-as {
    type uint32;
  }
  leaf peer-as {
    type uint32;
  }
  list peer {
    key name;
    leaf name {
      type ipaddress;
    }
    leaf local-as {
      type uint32;
      description
        ".... Defaults to ../local-as";
    }
    leaf peer-as {
      type uint32;
      description
        "... Defaults to ../peer-as";
    }
    leaf local-port {
      type inet:port;
    }
    leaf remote-port {
      type inet:port;
      default 179;
    }
    leaf state {
      config false;
      type enumeration {
        enum init;
        enum established;
        enum closing;
      }
    }
  }
}

```

In this example model, both `bgp/peer/local-as` and `bgp/peer/peer-as` have complex hierarchical values, allowing the user to specify default values for all peers in a single location.

The model also follows the pattern of fully integrating state ("config false") nodes with configuration ("config true") nodes. There is not separate "bgp-state" hierarchy, with the accompanying

repetition of containment and naming nodes. This makes the model simpler and more readable.

C.2.1. Datastores

Each datastore represents differing views of these nodes. <running> will hold the configuration provided by the user, for example a single BGP peer. <intended> will conceptually hold the data as validated, after the removal of data not intended for validation and after any local template mechanisms are performed. <operational> will show data from <intended> as well as any "config false" nodes.

C.2.2. Adding a Peer

If the user configures a single BGP peer, then that peer will be visible in both <running> and <intended>. It may also appear in <candidate>, if the server supports the "candidate" feature. Retrieving the peer will return only the user-specified values.

No time delay should exist between the appearance of the peer in <running> and <intended>.

In this scenario, we've added the following to <running>:

```
<bgp>
  <local-as>64642</local-as>
  <peer-as>65000</peer-as>
  <peer>
    <name>10.1.2.3</name>
  </peer>
</bgp>
```

C.2.2.1. <operational>

<operational> will contain the fully expanded peer data, including "config false" nodes. In our example, this means the "state" node will appear.

In addition, <operational> will contain the "currently in use" values for all nodes. This means that local-as and peer-as will be populated even if they are not given values in <intended>. The value of bgp/local-as will be used if bgp/peer/local-as is not provided; bgp/peer-as and bgp/peer/peer-as will have the same relationship. In the operational view, this means that every peer will have values for their local-as and peer-as, even if those values are not explicitly configured but are provided by bgp/local-as and bgp/peer-as.

Each BGP peer has a TCP connection associated with it, using the values of local-port and remote-port from <intended>. If those values are not supplied, the system will select values. When the connection is established, <operational> will contain the current values for the local-port and remote-port nodes regardless of the origin. If the system has chosen the values, the "origin" attribute will be set to "system". Before the connection is established, one or both of the nodes may not appear, since the system may not yet have their values.

```
<bgp origin="or:intended" xmlns="urn:example:bgp">
  <local-as origin="or:intended">64642</local-as>
  <peer-as origin="or:intended">65000</peer-as>
  <peer origin="or:intended">
    <name origin="or:intended">10.1.2.3</name>
    <local-as origin="or:default">64642</local-as>
    <peer-as origin="or:default">65000</peer-as>
    <local-port origin="or:system">60794</local-port>
    <remote-port origin="or:default">179</remote-port>
  </peer>
</bgp>
```

C.2.3. Removing a Peer

Changes to configuration may take time to percolate through the various software components involved. During this period, it is imperative to continue to give an accurate view of the working of the device. <operational> will contain nodes for both the previous and current configuration, as closely as possible tracking the current operation of the device.

Consider the scenario where a client removes a BGP peer. When a peer is removed, the operational state will continue to reflect the existence of that peer until the peer's resources are released, including closing the peer's connection. During this period, the current data values will continue to be visible in <operational>, with the "origin" attribute set to indicate the origin of the original data.

```

<bgp origin="or:intended">
  <local-as origin="or:intended">64642</local-as>
  <peer-as origin="or:intended">65000</peer-as>
  <peer origin="or:intended">
    <name origin="or:intended">10.1.2.3</name>
    <local-as origin="or:default">64642</local-as>
    <peer-as origin="or:default">65000</peer-as>
    <local-port origin="or:system">60794</local-port>
    <remote-port origin="or:default">179</remote-port>
  </peer>
</bgp>

```

Once resources are released and the connection is closed, the peer's data is removed from <operational>.

C.3. Interface Example

In this section, we'll use this simple interface data model:

```

container interfaces {
  list interface {
    key name;
    leaf name {
      type string;
    }
    leaf description {
      type string;
    }
    leaf mtu {
      type uint;
    }
    leaf ipv4-address {
      type inet:ipv4-address;
    }
  }
}

```

C.3.1. Pre-provisioned Interfaces

One common issue in networking devices is the support of Field Replaceable Units (FRUs) that can be inserted and removed from the device without requiring a reboot or interfering with normal operation. These FRUs are typically interface cards, and the devices support pre-provisioning of these interfaces.

If a client creates an interface "et-0/0/0" but the interface does not physically exist at this point, then <intended> might contain the following:

```
<interfaces>
  <interface>
    <name>et-0/0/0</name>
    <description>Test interface</description>
  </interface>
</interfaces>
```

Since the interface does not exist, this data does not appear in <operational>.

When a FRU containing this interface is inserted, the system will detect it and process the associated configuration. The <operational> will contain the data from <intended>, as well as the "config false" nodes, such as the current value of the interface's MTU.

```
<interfaces origin="or:intended">
  <interface origin="or:intended">
    <name origin="or:intended">et-0/0/0</name>
    <description origin="or:intended">Test interface</description>
    <mtu origin="or:system">1500</mtu>
  </interface>
</interfaces>
```

If the FRU is removed, the interface data is removed from <operational>.

C.3.2. System-provided Interface

Imagine if the system provides a loopback interface (named "lo0") with a default ipv4-address of "127.0.0.1". The system will only provide configuration for this interface if there is no data for it in <intended>.

When no configuration for "lo0" appears in <intended>, then <operational> will show the system-provided data:

```
<interfaces origin="or:intended">
  <interface origin="or:system">
    <name origin="or:system">lo0</name>
    <ipv4-address origin="or:system">127.0.0.1</ipv4-address>
  </interface>
</interfaces>
```

When configuration for "lo0" does appear in <intended>, then <operational> will show that data with the origin set to "intended". If the "ipv4-address" is not provided, then the system-provided value will appear as follows:

```
<interfaces origin="or:intended">
  <interface origin="or:intended">
    <name origin="or:intended">lo0</name>
    <description origin="or:intended">loopback</description>
    <ipv4-address origin="or:system">127.0.0.1</ipv4-address>
  </interface>
</interfaces>
```

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Juergen Schoenwaelder
Jacobs University

Email: j.schoenwaelder@jacobs-university.de

Phil Shafer
Juniper Networks

Email: phil@juniper.net

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Robert Wilton
Cisco Systems

Email: rwilton@cisco.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2017

M. Jethanandani
Cisco Systems, Inc
June 29, 2017

Accounting in NETCONF and RESTCONF
draft-mahesh-netconf-accounting-02

Abstract

This document defines an accounting record for NETCONF and RESTCONF.

Requirements Language

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].

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 December 31, 2017.

Copyright Notice

Copyright (c) 2017 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
1.1. Terminology	2
1.2. Compatability with remote AAA servers	3
2. Accounting Record	3
3. Data Model Definitions	4
3.1. Data Organization	4
3.2. YANG Module	5
4. IANA Considerations	10
5. Security Considerations	10
6. Acknowledgements	11
7. References	11
7.1. Normative References	11
7.2. Informative References	12
Author's Address	12

1. Introduction

NETCONF [RFC6241] and RESTCONF [RFC8040] protocol operations are authenticated and authorized as part of the Authentication, Authorization and Accounting (AAA) framework. An accounting record is generated as part of the same framework for each of these operations to satisfy the accounting part of AAA, but there has been no effort to define such a record. Having an accounting record that is consistent across vendors allows for the operator to compare operations across devices from different vendors. This document defines such a record and a corresponding YANG data model (ietf-netconf-am.yang).

The rest of this document will use NETCONF to imply both NETCONF and RESTCONF, but where applicable will call out each protocol specifically.

1.1. Terminology

The following terms are defined in NETCONF [RFC6241] and are not redefined here:

- o client
- o server
- o session

- o user
- o <get>
- o notification

1.2. Compatability with remote AAA servers

This document does not cover how the server interacts with remote AAA servers and any interaction is out of scope of this document. A particular implementation can make the records available as part of <get> request, send a notification every time a accounting record is generated or use any existing protocol to update the remote AAA server.

2. Accounting Record

An accounting record for NETCONF consists of the following fields. Note, there is no accounting record for reading or notification of an accounting record.

message-id	session-id	src-ip	date-time	user	groups	path	value	action	rule	status
------------	------------	--------	-----------	------	--------	------	-------	--------	------	--------

where:

message-id: This is the id within a given NETCONF session assigned to each RPC. RESTCONF has no concept of a session, so this field would be left blank.

session-id: The session-id in case of NETCONF and would be blank in case of RESTCONF. If the accounting record needs to be fragmented for any reason, it is suggested that this field not be repeated in subsequent packets. Instead a combination of start and end record marker, and the message-id should be used to reassemble fragmented records.

src-ip: The source IP address that was used to request the operation. If the accounting record needs to be fragmented for any reason, it is suggested that this field not be repeated in subsequent packets. Instead a combination of start and end record marker, and the message-id should be used to reassemble fragmented records.

date-time: The date and time when the operation was performed (UTC Timezone). If the accounting record needs to be fragmented for any reason, it is suggested that this field not be repeated in subsequent packets. Instead a combination of start and end record marker, and the message-id should be used to reassemble fragmented records.

user: The NETCONF user that requested this operation. If the accounting record needs to be fragmented for any reason, it is suggested that this field not be repeated in subsequent packets. Instead a combination of start and end record marker, and the message-id should be used to reassemble fragmented records.

groups: The group the user belongs to. If the accounting record needs to be fragmented for any reason, it is suggested that this field not be repeated in subsequent packets. Instead a combination of start and end record marker, and the message-id should be used to reassemble fragmented records.

path: The path in the NACM [RFC6536] rule on which the operations is being performed

value: The value that was set for any of the attributes in the request

action: The action in the NACM [RFC6536] rule

rule: The rule in the NACM [RFC6536] that was used to authorize the action.

status: Whether the operations was permitted or denied.

3. Data Model Definitions

The model uses the NACM extension statement of default-deny-all to protect accounting records. Explicit rules have to be defined to be enable access to the accounting records.

3.1. Data Organization

The following diagram highlights the contents and structure of the Accounting YANG module. For information on annotations, please refer to YANG Tree Diagrams [I-D.ietf-netmod-yang-tree-diagrams].

```
module: ietf-netconf-am
  +--rw nam
    +--rw enable-nam?          boolean
    +--rw read-default?        nacm:action-type
    +--ro accounting-record* [session-id message-id]
      +--ro session-id        nc:session-id-type
      +--ro message-id        uint32
      +--ro date-time          yang:date-and-time
      +--ro src-ip             inet:ip-address
      +--ro group              nacm:group-name-type
      +--ro user?              nacm:user-name-type
      +--ro path               nacm:node-instance-identifier
      +--ro value?
      +--ro action             nacm:access-operations-type
      +--ro rule?              string
      +--ro status?            nacm:action-type
```

3.2. YANG Module

The following YANG module specifies the normative NETCONF content that MUST be supported by the server.

The "ietf-netconf-am" YANG module imports typedefs from YANG-TYPES [RFC6991], from NETCONF [RFC6241] and from NACM [RFC6536].

<CODE BEGINS> file "ietf-netconf-am@2017-06-29.yang"

```
module ietf-netconf-am {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-am";
  prefix "nam";

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-yang-types {
    prefix yang;
  }

  import ietf-netconf {
    prefix nc;
  }

  import ietf-netconf-acm {
    prefix nacm;
  }
}
```

organization

"IETF NETCONF (Network Configuration) Working Group";

contact

"WG Web: <<http://tools.ietf.org/wg/netconf/>>
WG List: <<mailto:netconf@ietf.org>>

WG Chair: Mehmet Ersue
<<mailto:mehmet.ersue@nsn.com>>

WG Chair: Mahesh Jethanandani
<<mailto:mjethanandani@gmail.com>>

Editor: Mahesh Jethanandani
<<mailto:mjethanandani@gmail.com>>"

description

"This module defines an accounting record for NETCONF operations performed on the server. If these operations are authorized using rules defined by NACM [RFC6536], then that information is also captured by this module.

Copyright (c) 2014 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices."

```
revision "2017-06-29" {  
  description  
    "Initial version";  
  reference  
    "RFC XXXX: NETCONF and RESTCONF Accounting";  
}
```

```
/*  
 * Data definition statements.  
 */
```

```
container nam {
```

```
nacm:default-deny-all;

description
  "Parameters for NETCONF Accounting Model.";

leaf enable-nam {
  type boolean;
  default true;
  description
    "Enable or disable generation of NETCONF
    accounting records. If 'true', accounting
    records will be generated. If set to 'false'
    no accounting records will be generated.";
}

leaf read-default {
  type nacm:action-type;
  default "permit";
  description
    "Controls whether read access is granted if
    no particular rule is found for a read
    request.";
}

list accounting-record {
  key "session-id message-id";
  config false;
  description
    "A list of accounting records generated by the server";

  leaf session-id {
    type nc:session-id-type;
    description
      "If this operation happened over NETCONF, this
      field captures the NETCONF session-id. In case
      of RESTCONF this field can be left blank.";
  }

  leaf message-id {
    type uint32;
    description
      "Id that is assigned to each RPC within a given
      NETCONF session. Should be blank in case of
      RESTCONF.";
  }

  leaf date-time {
    type yang:date-and-time;
  }
}
```

```
    mandatory true;
    description
        "The date and time when the operation was
        requested.";
}

leaf src-ip {
    type inet:ip-address;
    mandatory true;
    description
        "The source IP address where the request was made
        from.";
}

leaf group {
    type nacm:group-name-type;
    mandatory true;
    description
        "The name of the group that the user who requested
        the operation belongs to.";
}

leaf user {
    type nacm:user-name-type;
    description
        "The user within the group that is requesting this
        operation.";
}

leaf path {
    type nacm:node-instance-identifier;
    mandatory true;
    description
        "Data Node Instance Identifier associated with the
        data node that the request is being made on.

        Instance identifiers start with the top-level
        data node, and a complete identifier is required
        for this value.";
}

anydata value {
    description
        "An optional field, it contains the value of any
        of the attribute that form the record.

        It could be as simple as the filter value
        'http' specified that the user requested as part
```

of the authorization request such as in this example:

```
<filter>
  <name>http</name>
</filter>
```

or it could be value being set for a ssh port in this example:

```
<ssh>
  <port>2022</port>
</ssh>"
```

```
}
leaf action {
  type nacm:access-operations-type;
  mandatory true;
  description
    "The type of NETCONF operation being requested.";
}

leaf rule {
  type string {
    length "1..max";
  }
  description
    "The name assigned to the rule that was used to
    authorize the action, if authorization was
    enabled.";
}

leaf status {
  type nacm:action-type;
  description
    "Action taken by the server when the above
    mentioned rule matched, if authorization was
    enable.";
}
}
}
```

<CODE ENDS>

4. IANA Considerations

This document makes two requests of IANA.

The first request is to register one URI in "The IETF XML Registry". Following the format in The IETF XML Registry [RFC3688], the following needs to be registered.

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-am

Registrant Contact: The IESG

XML: N/A, the requested URI is an XML namespace

The second request is to register one module in the "YANG Module Names" registry. Following the format in YANG [RFC7950], the following needs to be registered.

Name: ietf-netconf-am

Namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-am

Prefix: nam

Reference: RFC XXXX

Note to RFC Editor - Please replace XXXX here and in the rest of the draft with the RFC id assigned to this draft.

5. Security Considerations

The YANG module defined in this document is designed to be accessed via network management protocol such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layers is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF Access Control Model (NACM) [RFC6536] provides the means to restrict access for particular NETCONF or RESTCONF users to a pre-configured subset of all available NETCONF or RESTCONF protocol operations and content.

Most of the data nodes defined in this YANG module are readonly, i.e. config false, and are therefore not vulnerable to manipulation in network environments. However, they might contain data that might be sensitive and should be protected with the right NACM [RFC6536] rules.

6. Acknowledgements

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<http://www.rfc-editor.org/info/rfc3688>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [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>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<http://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

7.2. Informative References

[I-D.ietf-netmod-yang-tree-diagrams]

Bjorklund, M. and L. Berger, "YANG Tree Diagrams", draft-ietf-netmod-yang-tree-diagrams-00 (work in progress), June 2017.

Author's Address

Mahesh Jethanandani
Cisco Systems, Inc
170 West Tasman Drive
San Jose, CA 95070
USA

Email: mjethanandani@gmail.com

Network Working Group
Internet-Draft
Obsoletes: rfc7895 (if approved)
Updates: rfc7950, rfc8040 (if approved)
Intended status: Standards Track
Expires: January 4, 2018

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
K. Watsen
Juniper Networks
July 3, 2017

YANG Library
draft-nmdsdt-netconf-rfc7895bis-01

Abstract

This document describes a YANG library that provides information about all the YANG modules used by a network management server (e.g., a Network Configuration Protocol (NETCONF) server). Simple caching mechanisms are provided to allow clients to minimize retrieval of this information.

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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 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
1.1. Terminology	3
1.2. Tree Diagrams	4
1.3. Motivation for rfc7895bis	4
1.4. Summary of Changes from RFC 7895	5
1.5. Summary of Updates to RFC 7950	6
1.6. Summary of Updates to RFC 8040	6
1.7. Open Issues	6
2. YANG Library	7
2.1. yang-library	9
2.1.1. yang-library/modules/module	9
2.1.2. yang-library/module-sets/module-set	9
2.1.3. yang-library/datastores/datastore	9
2.2. modules-state	9
2.2.1. modules-state/module-set-id	9
2.2.2. modules-state/module	10
2.3. YANG Library Module	10
3. IANA Considerations	20
3.1. YANG Module Registry	20
4. Security Considerations	21
5. Acknowledgements	21
6. References	21
6.1. Normative References	21
6.2. Informative References	23
Authors' Addresses	23

1. Introduction

There is a need for standard mechanisms to provide the operational state of the server. This includes, for instance, identifying the YANG modules and datastores that are in use by a server and how they relate to each other.

If a large number of YANG modules are utilized by the server, then the YANG library contents needed can be relatively large. This information changes very infrequently, so it is important that clients be able to cache the YANG library contents and easily identify whether their cache is out of date.

YANG library information can be different on every server and can change at runtime or across a server reboot.

If the server implements multiple protocols to access the YANG-defined data, each such protocol has its own conceptual instantiation of the YANG library.

The following information is needed by a client application (for each YANG module in the library) to fully utilize the YANG data modeling language:

- o identifier: a unique identifier for the module that includes the module's name, revision, features, and deviations.
- o name: The name of the YANG module.
- o revision: Each YANG module and submodule within the library has a revision. This is derived from the most recent revision statement within the module or submodule. If no such revision statement exists, the module's or submodule's revision is the zero-length string.
- o submodule list: The name and revision of each submodule used by the module MUST be identified.
- o feature list: The name of each YANG feature supported by the server, in a given context, MUST be identified.
- o deviation list: The name of each YANG module used for deviation statements, in a given context, MUST be identified.

The following information is needed by a client application (for each datastore supported by the server) to fully access all the YANG-modelled data available on the server:

- o identity: the YANG identity for the datastore.
- o properties: properties supported by the datastore.
- o modules: modules supported by the datastore, including any features and deviations.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

The following terms are defined in [RFC6241]:

- o client
- o server

The following terms are defined in [RFC7950]:

- o module
- o submodule

The following terms are used within this document:

- o YANG library: A collection of metadata describing the server's operational state.

1.2. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

1.3. Motivation for rfc7895bis

RFC Ed.: delete this section, including this note, at time of publication.

All NETCONF servers supporting YANG 1.1 [RFC7950] MUST support YANG Library (see Section 5.6.4 of RFC 7950). Similarly, all RESTCONF servers MUST support YANG Library (see Section 10 of RFC 8040). These requirements are independent of if the server supports NMDA or not.

RFC 7895 has a mandatory to implement 'modules-state' tree that a server uses to advertise all the modules it supports. However, this

module was designed assuming the all modules would be in all datastores, and with the same number of features and deviations. However, this is not the case with NMDA-compatible servers that may have some modules that only appear in <operational> (e.g., ietf-network-topo) or only also appear in a dynamic datastore (e.g., i2rs-ephemeral-rib). It is also possible that a server only implements a module in <running>, at it hasn't yet coded support for returning the module's opstate yet. Presumably, an NMDA-supporting server would return all modules implemented in every datastore, but this would be misleading to existing clients and unhelpful to NMDA-aware clients.

In the end, it appears that the 'modules-state' node should be for non-NMDA aware clients. For backwards compatability, an NMDA-supporting server SHOULD populate 'modules-state' in a backwards-compatible manner. The new 'yang-library' node would be ignored by legacy clients, while providing all the data needed for NMDA-aware clients, which would themselves ignore the 'modules-state' tree.

In addition to resolving the 'modules-state' node NMDA-incompatibility issue described above, the solution presented in this document is further motivated by the following desires:

- o leverage Section 5.6.4 of RFC 7950 and Section 10 of RFC 8040.
- o indicate which modules are supported by each datastore
- o enable the features and deviations to vary by datastore
- o structure extensible to support schema-mount
- o provide a top-level container for all server metadata

1.4. Summary of Changes from RFC 7895

This document updates [RFC7895] in the following ways:

- o Renames document title from "YANG Module Library" to "YANG Library".
- o Adds new top-level "yang-library" container to hold many types of server metadata: modules supported, datastores supported, relationships between datastores and modules, etc.
- o Deprecates the modules-state tree.

1.5. Summary of Updates to RFC 7950

This document updates [RFC7950] in the following ways:

1. Section 5.6.4 says:

A NETCONF server MUST announce the modules it implements (see Section 5.6.5) by implementing the YANG module "ietf-yang-library" defined in [RFC7895] and listing all implemented modules in the "/modules-state/module" list.

This should be updated to allow for also listing all implemented modules in the "/yang-library/modules/module" list or, more generally, use the entire YANG Library.

2. Section 5.6.4 also says:

The parameter "module-set-id" has the same value as the leaf "/modules-state/module-set-id" from "ietf-yang-library". This parameter MUST be present.

This should be updated to say that, for NMDA-capable servers, this parameter has the same value as the leaf "/yang-library/module-sets/module-set/id", for the module-set that is used by <running>.

1.6. Summary of Updates to RFC 8040

This document updates [RFC8040] in the following ways:

1. Section 10.1 says that the "modules-state/module" list is mandatory. This should be updated to allow for also listing all supported modules in the "/yang-library/modules/module" list or, more generally, use the entire YANG Library.

1.7. Open Issues

- o The per-datastore 'properties' idea is still being discussed. It's included here so as to provide something to point at.
- o There's debate if there should be a list of module-sets or if instead each 'module-set' should be embedded into the datastore definition. This discussion goes into if a datastore can reference more than one module-set.

2. YANG Library

The "ietf-yang-library" module provides information about the modules used by a server. This module is defined using YANG version 1, but it supports the description of YANG modules written in any revision of YANG.

Following is the YANG Tree Diagram for the "ietf-yang-library" module, including the deprecated 'modules-state' tree:

```

+--ro yang-library
|
|  +--ro modules
|  |
|  |  +--ro module* [id]
|  |  |
|  |  |  +--ro id          string
|  |  |  +--ro name?       yang:yang-identifier
|  |  |  +--ro revision?   union
|  |  |  +--ro schema?     inet:uri
|  |  |  +--ro namespace   inet:uri
|  |  |  +--ro feature*    yang:yang-identifier
|  |  |  +--ro deviation* [name revision]
|  |  |  |  +--ro name      yang:yang-identifier
|  |  |  |  +--ro revision  union
|  |  |  +--ro conformance-type enumeration
|  |  |  +--ro submodule* [name revision]
|  |  |  |  +--ro name      yang:yang-identifier
|  |  |  |  +--ro revision  union
|  |  |  |  +--ro schema?   inet:uri
|  |
|  |  +--ro module-sets
|  |  |
|  |  |  +--ro module-set*
|  |  |  |
|  |  |  |  +--ro id?      string
|  |  |  |  +--ro module*  -> /yang-library/modules/module/id
|  |
|  |  +--ro datastores
|  |  |
|  |  |  +--ro datastore* [name]
|  |  |  |
|  |  |  |  +--ro name      identityref
|  |  |  |  +--ro properties
|  |  |  |  |  +--ro property* identityref
|  |  |  |  +--ro module-set? -> /yang-library/module-sets/module-set/id
|
|  x--ro modules-state
|  |
|  |  +--ro module-set-id  string
|  |  +--ro module* [name revision]
|  |  |
|  |  |  +--ro name          yang:yang-identifier
|  |  |  +--ro revision      union
|  |  |  +--ro schema?      inet:uri
|  |  |  +--ro namespace    inet:uri
|  |  |  +--ro feature*     yang:yang-identifier
|  |  |  +--ro deviation* [name revision]
|  |  |  |  +--ro name      yang:yang-identifier
|  |  |  |  +--ro revision  union
|  |  |  +--ro conformance-type enumeration
|  |  |  +--ro submodule* [name revision]
|  |  |  |  +--ro name      yang:yang-identifier
|  |  |  |  +--ro revision  union
|  |  |  |  +--ro schema?   inet:uri

```

2.1. yang-library

This mandatory container holds all of the server's metadata.

2.1.1. yang-library/modules/module

This mandatory list contains one entry for each unique instance of a module in use by the server. Each entry is distinguished by the module's name, revisions, features, and deviations.

2.1.2. yang-library/module-sets/module-set

This mandatory list contains one entry for each module-set in use by the server (e.g., presented by a datastore).

2.1.3. yang-library/datastores/datastore

This mandatory list contains one entry for each datastore supported by the server. Each datastore entry both identifies any special properties it has and any module-sets it uses.

2.2. modules-state

This mandatory container holds the identifiers for the YANG data model modules supported by the server.

2.2.1. modules-state/module-set-id

This mandatory leaf contains a unique implementation-specific identifier representing the current set of modules and submodules on a specific server. The value of this leaf MUST change whenever the set of modules and submodules in the YANG library changes. There is no requirement that the same set always results in the same "module-set-id" value.

This leaf allows a client to fetch the module list once, cache it, and only refetch it if the value of this leaf has been changed.

If the value of this leaf changes, the server also generates a "yang-library-change" notification, with the new value of "module-set-id".

Note that for a NETCONF server that implements YANG 1.1 [RFC7950], a change of the "module-set-id" value results in a new value for the :yang-library capability defined in [RFC7950]. Thus, if such a server implements NETCONF notifications [RFC5277], and the notification "netconf-capability-change" [RFC6470], a

"netconf-capability-change" notification is generated whenever the "module-set-id" changes.

2.2.2. modules-state/module

This mandatory list contains one entry for each YANG data model module supported by the server. There MUST be an entry in this list for each revision of each YANG module that is used by the server. It is possible for multiple revisions of the same module to be imported, in addition to an entry for the revision that is implemented by the server.

2.3. YANG Library Module

The "ietf-yang-library" module defines monitoring information for the YANG modules used by a server.

The modules "ietf-yang-types" and "ietf-inet-types" from [RFC6991] and the module "ietf-datastores" from [I-D.ietf-netmod-revised-datastores] are used by this module for some type definitions.

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-yang-library@2017-07-03.yang"

```
module ietf-yang-library {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-library";
  prefix "yanglib";

  import ietf-yang-types {
    prefix yang;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-datastores {
    prefix ds;
    reference "I-D.ietf-revised-datastores:
              Network Management Datastore Architecture.";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";
```

contact

"WG Web: <<http://tools.ietf.org/wg/netconf/>>
WG List: <<mailto:netconf@ietf.org>>

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

Author: Martin Bjorklund
<<mailto:mbj@tail-f.com>>

Author: Kent Watsen
<<mailto:kwatsen@juniper.net>>;

description

"This module contains monitoring information about the YANG modules and submodules that are used within a YANG-based server.

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

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

```
// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
// RFC Ed.: replace XXXX with actual RFC number and remove this
// note.
revision 2017-07-03 {
  description
    "Updated revision.";
  reference
    "RFC XXXX: YANG Library.";
}
revision 2016-04-09 {
  description
    "Initial revision.";
  reference
    "RFC 7895: YANG Module Library.";
}
```

```
/*
 * Typedefs
 */

typedef revision-identifier {
  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }
  description
    "Represents a specific date in YYYY-MM-DD format.";
}

/*
 * Groupings
 */
grouping common-leafs2 {
  description
    "Common parameters for YANG modules and submodules.";

  leaf name {
    type yang:yang-identifier;
    description
      "The YANG module or submodule name.";
  }
  leaf revision {
    type union {
      type revision-identifier;
      type string { length 0; }
    }
    description
      "The YANG module or submodule revision date.
      A zero-length string is used if no revision statement
      is present in the YANG module or submodule.";
  }
}

grouping schema-leaf2 {
  description
    "Common schema leaf parameter for modules and submodules.";

  leaf schema {
    type inet:uri;
    description
      "Contains a URL that represents the YANG schema
      resource for this module or submodule.

      This leaf will only be present if there is a URL
```

```
        available for retrieval of the schema for this entry.";
    }
}

/*
 * Top-level container
 */
container yang-library {
    config false;
    description
        "Top-level resource providing all the meta information the
        server possesses.";

    container modules {
        description
            "A container holding a list of modules. Note, modules being
            listed here does not mean that they are supported by any
            particular datastore.";

        list module {
            key "id";

            description
                "Each entry represents one revision of one module
                currently supported by the server.";

            leaf id {
                type string;
                description
                    "A system-generated value that uniquely represents the
                    module listing, including its name, revision, features,
                    and deviations.";
            }

            uses common-leafs2;
            uses schema-leaf2;

            leaf namespace {
                type inet:uri;
                mandatory true;
                description
                    "The XML namespace identifier for this module.";
            }

            leaf-list feature {
                type yang:yang-identifier;
                description
                    "List of YANG feature names from this module that are
```



```
        supported by the server, regardless whether they are
        defined in the module or any included submodule.";
    }
    list deviation {
        key "name revision";
        description
            "List of YANG deviation module names and revisions
            used by this server to modify the conformance of
            the module associated with this entry. Note that
            the same module can be used for deviations for
            multiple modules, so the same entry MAY appear
            within multiple 'module' entries.

            The deviation module MUST be present in the 'module'
            list, with the same name and revision values.
            The 'conformance-type' value will be 'implement' for
            the deviation module.";
        uses common-leafs2;
    }
    leaf conformance-type {
        type enumeration {
            enum implement {
                description
                    "Indicates that the server implements one or more
                    protocol-accessible objects defined in the YANG module
                    identified in this entry. This includes deviation
                    statements defined in the module.

                    For YANG version 1.1 modules, there is at most one
                    module entry with conformance type 'implement' for a
                    particular module name, since YANG 1.1 requires that
                    at most one revision of a module is implemented.

                    For YANG version 1 modules, there SHOULD NOT be more
                    than one module entry for a particular module name.";
            }
        }
        enum import {
            description
                "Indicates that the server imports reusable definitions
                from the specified revision of the module, but does
                not implement any protocol accessible objects from
                this revision.

                Multiple module entries for the same module name MAY
                exist. This can occur if multiple modules import the
                same module, but specify different revision-dates in
                the import statements.";
        }
    }
```

```
    }
    mandatory true;
    description
        "Indicates the type of conformance the server is claiming
        for the YANG module identified by this entry.";
    }
    list submodule {
        key "name revision";
        description
            "Each entry represents one submodule within the
            parent module.";
        uses common-leafs2;
        uses schema-leaf2;
    }
}

container module-sets {
    description
        "A container for a list of module-sets. Module-sets being
        listed here does not mean that they are used by any
        particular datastore.";
    list module-set {
        description
            "An arbitrary module-set definition provided by the server.";

        leaf id {
            type string;
            description
                "A server-supplied identifier for this set of modules.";
        }
        leaf-list module {
            type leafref {
                path "/yang-library/modules/module/id";
            }
            description
                "A module-instance supported by the server, including its
                features and deviations.";
        }
    }
}

container datastores {
    description
        "A container for a list of datastores supported by the server.
        Each datastore indicates which module-sets it supports.";

    list datastore {
```

```
    key name;
    leaf name {
        type identityref {
            base ds:datastore;
        }
        description
            "The identity of the datastore.";
    }
    container properties {
        leaf-list property {
            type identityref {
                base ds:property;
            }
            description
                "A property of the datastore.";
        }
        description
            "A list of properties supported by this datastore.";
    }
    leaf module-set {
        type leafref {
            path "/yang-library/module-sets/module-set/id";
        }
        description
            "A reference to a module-set supported by this datastore";
    }
    description
        "A datastore supported by this server.";
} // end 'datastores'

} // end 'yang-library'

/*
 * Legacy groupings
 */

grouping module-list {
    description
        "The module data structure is represented as a grouping
        so it can be reused in configuration or another monitoring
        data structure.";

    grouping common-leafs {
        description
            "Common parameters for YANG modules and submodules.";
    }
}
```

```
    leaf name {
      type yang:yang-identifier;
      description
        "The YANG module or submodule name.";
    }
    leaf revision {
      type union {
        type revision-identifier;
        type string { length 0; }
      }
      description
        "The YANG module or submodule revision date.
        A zero-length string is used if no revision statement
        is present in the YANG module or submodule.";
    }
  }
}

grouping schema-leaf {
  description
    "Common schema leaf parameter for modules and submodules.";

  leaf schema {
    type inet:uri;
    description
      "Contains a URL that represents the YANG schema
      resource for this module or submodule.

      This leaf will only be present if there is a URL
      available for retrieval of the schema for this entry.";
  }
}

list module {
  key "name revision";
  description
    "Each entry represents one revision of one module
    currently supported by the server.";

  uses common-leafs;
  uses schema-leaf;

  leaf namespace {
    type inet:uri;
    mandatory true;
    description
      "The XML namespace identifier for this module.";
  }
  leaf-list feature {
```

```
type yang:yang-identifier;
description
  "List of YANG feature names from this module that are
   supported by the server, regardless whether they are
   defined in the module or any included submodule.";
}
list deviation {
  key "name revision";
  description
    "List of YANG deviation module names and revisions
     used by this server to modify the conformance of
     the module associated with this entry. Note that
     the same module can be used for deviations for
     multiple modules, so the same entry MAY appear
     within multiple 'module' entries.

     The deviation module MUST be present in the 'module'
     list, with the same name and revision values.
     The 'conformance-type' value will be 'implement' for
     the deviation module.";
  uses common-leafs;
}
leaf conformance-type {
  type enumeration {
    enum implement {
      description
        "Indicates that the server implements one or more
         protocol-accessible objects defined in the YANG module
         identified in this entry. This includes deviation
         statements defined in the module.

         For YANG version 1.1 modules, there is at most one
         module entry with conformance type 'implement' for a
         particular module name, since YANG 1.1 requires that
         at most one revision of a module is implemented.

         For YANG version 1 modules, there SHOULD NOT be more
         than one module entry for a particular module name.";
    }
  }
  enum import {
    description
      "Indicates that the server imports reusable definitions
       from the specified revision of the module, but does
       not implement any protocol accessible objects from
       this revision.

       Multiple module entries for the same module name MAY
       exist. This can occur if multiple modules import the
```

```
        same module, but specify different revision-dates in
        the import statements.";
    }
}
mandatory true;
description
    "Indicates the type of conformance the server is claiming
    for the YANG module identified by this entry.";
}
list submodule {
    key "name revision";
    description
        "Each entry represents one submodule within the
        parent module.";
    uses common-leafs;
    uses schema-leaf;
}
}
}

/*
 * Legacy operational state data nodes
 */

container modules-state {
    config false;
    status deprecated;
    description
        "Contains YANG module monitoring information.";

    leaf module-set-id {
        type string;
        mandatory true;
        description
            "Contains a server-specific identifier representing
            the current set of modules and submodules. The
            server MUST change the value of this leaf if the
            information represented by the 'module' list instances
            has changed.";
    }

    uses module-list;
}

/*
 * Notifications
 */
```

```
notification yang-library-change {
  description
    "Generated when the set of modules and submodules supported
    by the server has changed.";
  leaf module-set-id {
    type leafref {
      path "/yanglib:modules-state/yanglib:module-set-id";
    }
    mandatory true;
    description
      "Contains the module-set-id value representing the
      set of modules and submodules supported at the server at
      the time the notification is generated.";
  }
}
}
```

<CODE ENDS>

3. IANA Considerations

3.1. YANG Module Registry

RFC 7895 previously registered one URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration was made:

URI: urn:ietf:params:xml:ns:yang:ietf-yang-library
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

This document takes over this registration entry made by RFC 7895.

RFC 7895 previously registered one YANG module in the "YANG Module Names" registry [RFC6020] as follows:

name:	ietf-yang-library
namespace:	urn:ietf:params:xml:ns:yang:ietf-yang-library
prefix:	yanglib
reference:	RFC 7895

This document takes over this registration entry made by RFC 7895.

4. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- o /modules-state/module: The module list used in a server implementation may help an attacker identify the server capabilities and server implementations with known bugs. Although some of this information may be available to all users via the NETCONF <hello> message (or similar messages in other management protocols), this YANG module potentially exposes additional details that could be of some assistance to an attacker. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. This information is included in each module entry. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the module list information will help an attacker identify server implementations with such a defect, in order to launch a denial-of-service attack on the device.

5. Acknowledgements

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

6. References

6.1. Normative References

- [I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore
Architecture", draft-ietf-netmod-revised-datastores-03
(work in progress), July 2017.
- [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>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
DOI 10.17487/RFC3688, January 2004,
<<http://www.rfc-editor.org/info/rfc3688>>.
- [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>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure
Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011,
<<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration
Protocol (NETCONF) Access Control Model", RFC 6536,
DOI 10.17487/RFC6536, March 2012,
<<http://www.rfc-editor.org/info/rfc6536>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types",
RFC 6991, DOI 10.17487/RFC6991, July 2013,
<<http://www.rfc-editor.org/info/rfc6991>>.
- [RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module
Library", RFC 7895, DOI 10.17487/RFC7895, June 2016,
<<http://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,
<<http://www.rfc-editor.org/info/rfc7950>>.

- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<http://www.rfc-editor.org/info/rfc8040>>.

6.2. Informative References

- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<http://www.rfc-editor.org/info/rfc5277>>.
- [RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <<http://www.rfc-editor.org/info/rfc6470>>.

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2018

E. Voit
Cisco Systems
A. Bierman
YumaWorks
A. Clemm
Huawei
T. Jenkins
Cisco Systems
July 2, 2017

Notification Message Headers and Bundles
draft-voit-netconf-notification-messages-01

Abstract

This document specifies transport independent capabilities for messages transporting event notifications and YANG datastore update records. Included are:

- o a set of transport agnostic message header objects, and
- o how to associate a subset of these header objects with one or more events, YANG datastore updates, and/or alarms.

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 January 3, 2018.

Copyright Notice

Copyright (c) 2017 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	3
3. Header Objects	3
4. Transport independent headers for notifications	4
5. Bundled Notifications	6
6. Data Model	7
7. Security Considerations	13
8. References	13
8.1. Normative References	13
8.2. Informative References	13
Appendix A. Issues being worked	14
Appendix B. Querying an Object Model	14
Authors' Addresses	15

1. Introduction

Mechanisms to support subscription to event notifications and yang datastore push are being defined in [subscribe] and [yang-push]. Work on those documents have shown that existing YANG notifications described in [RFC7950] section 7.16 do not expose some useful transport independent capabilities that application developers are requesting. Communicating information on the following objects should not require knowledge of the underlying transport:

- o the kind of information encapsulated (event, data objects, alarm?)
- o the time information was generated
- o the time the information was sent
- o a signature to verify authenticity
- o the process generating the information
- o an originating request correlation

- o an ability to bundle information records together in a message
- o the ability to check for message loss/reordering

The document describes information elements needed for the functions above. It also provides YANG Notifications for sending messages containing one or more events and/or update records to a receiver.

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].

Definitions of Notification, Event, Event Notification, Receiver, and Subscription are defined in [subscribe].

3. Header Objects

There are a number of transport independent headers which should have common definition across applications. These include:

- o record-type: defines the kind of information assembled as a unit. (E.g., is it a YANG datastore update, an alarm, an event, etc.)
- o subscription-id: provides a reference into the reason the originator believed the receiver wishes to be notified of this specific information.
- o record-time: the time an event, datastore update, or other item it itself is recognized in the originating system.
- o record-id: identifies an event notification on an originator.
- o observation-domain-id: identifies the originator process which discovered and recorded the event notification. (note: look to reuse the domains set up with IPFIX.)
- o notification-time: the time the message was packaged sent to the transport layer for delivery to the receiver.
- o signature: allows an application to sign a message so that a receiver can verify the authenticity of the message.
- o dscp: network qos encoding which an application suggests should be applied to the message.

- o notification-id: for a specific message generator, this identifies a message which includes one or more event records.
- o previous-notification-id: the notification-id of the message preceding the current one intended for the same receiver. When used in conjunction with the current notification-id, this allows loss/duplication across previous messages to be discovered. If there was no previous message from a message generator, the reserved id "0" must be sent.
- o message-generator-id: identifier for the process which created the message notification. This allows disambiguation of an information source, such as the identification of different line cards sending the notification messages. Used in conjunction with previous-notification-id, this can help find drops and duplications when notifications are coming from multiple sources on a device. If there is a message-generator-id in the header, then the previous-notification-id should be the notification-id from the last time that message-generator-id was sent.

4. Transport independent headers for notifications

Many objects may be placed within a notification. However only a certain subset these objects are of potential use to networking layers prior the notification being interpreted by some receiving application layer process. By exposing this object information as part of a header, and by using standardized object names, it becomes possible for this object information to be leveraged in transit.

The objects defined in the previous section effectively become well-known objects where, if in the header, may act as supplemental information in communications between two devices. These well-known header fields are encapsulated within a dedicated subtree which leads off the notification message. This allows header objects to be easily be decoupled, stripped, and processed separately.

Typically sequence of information in YANG models is irrelevant. But as part of a transported notification, It is useful to sequence these header objects so that processing is as efficient as possible. This allows the handling or discarding of uninteresting notifications quickly.

Below is are record objects contents would include the objects presented in the section above. The proper way this message would be generated would be to look for the well known object names and place them in the header. All other would be placed in the notification record contents. (Note: are there any of these we should rather duplicate than move?)

```

+---n notification-message
  +---ro notification-message-header
    |   +---ro record-time           yang:date-and-time
    |   +---ro record-type?         notification-record-format
    |   +---ro subscription-id*      uint32
    |   +---ro record-id?           uint32
    |   +---ro observation-domain-id? string
    |   +---ro notification-id?     uint32
    |   +---ro notification-time?   yang:date-and-time
    |   +---ro previous-notification-id? uint32
    |   +---ro dscp?                inet:dscp
    |   +---ro message-generator-id? string
    |   +---ro signature?           string
  +---ro receiver-record-contents?

```

An actual instance of a notification might look like:

```

<notification
  xmlns="urn:ietf:params:xml:ns:netmod:notification:2.0">
  <notification-message-header>
    <record-time>
      2017-02-14T00:00:02Z
    </record-time>
    <record-type>
      yang-patch
    </record-type>
    <subscription-identifier>
      823472
    </subscription-identifier>
    <notification-time>
      2017-02-14T00:00:05Z
    </notification-time>
    <notification-identifier>
      456
    </notification-identifier>
    <previous-notification-identifier>
      567
    </previous-notification-identifier>
    <signature>
      lKIo8s03fd23.....
    </signature>
  </notification-message-header>
  <datastore-changes>
    ...(yang patch here)...
  </datastore-changes>
</notification>

```

5. Bundled Notifications

In many implementations, it may be inefficient to transport every notification independently. Instead, scale and processing speed can be improved by placing multiple notifications into one transportable bundle.

When this is done, one additional header field becomes valuable. This is the "record-count" which would tally the quantity of records which make up the contents of the bundle.

The format of a bundle would look as below. When compared to the unbundled notification, note that the headers have been split so that one set of headers associated with the notification occur once at the beginning of the message, and additional record specific headers which occur before individual records.

```

+---n bundled-notification-message
  +---ro bundled-notification-message-header
    |   +---ro notification-id?          uint32
    |   +---ro notification-time         yang:date-and-time
    |   +---ro previous-notification-id? uint32
    |   +---ro dscp?                    inet:dscp
    |   +---ro message-generator-id?    string
    |   +---ro signature?               string
    |   +---ro record-count?            uint16
  +---ro notification-records*
    +---ro notification-record-header
      |   +---ro record-time             yang:date-and-time
      |   +---ro record-type?            notification-record-format
      |   +---ro subscription-id*        uint32
      |   +---ro record-id?              uint32
      |   +---ro observation-domain-id?  string
    +---ro receiver-record-contents?

```

An actual instance of a bundled notification might look like:


```

<notification
  xmlns="urn:ietf:params:xml:ns:netmod:notification:2.0">
  <bundled-notification-message-header>
    <notification-time>
      2017-02-14T00:00:05Z
    </notification-time>
    <notification-identifier>
      456
    </notification-identifier>
    <previous-notification-identifier>
      567
    </previous-notification-identifier>
    <signature>
      lKIo8s03fd23...
    </signature>
    <record-count>
      2
    </record-count>
  </bundled-notification-message-header>
  <notification-record>
    <notification-record-header>
      <record-time>
        2017-02-14T00:00:02Z
      </record-time>
      <record-type>
        yang-patch
      </record-type>
      <subscription-identifier>
        823472
      </subscription-identifier>
    </notification-record-header>
    <notification
      xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
      <datastore-changes>
        ...(yang patch here)...
      </datastore-changes>
    </notification>
  </notification-record>
  <notification-record>
    ...(record #2)...
  </notification-record>
</notification>

```

6. Data Model

```

<CODE BEGINS> file "ietf-notification-messages.yang"
module ietf-notification-messages {
  yang-version 1.1;

```

```
namespace "urn:ietf:params:xml:ns:yang:ietf-notification-messages";
prefix nm;

import ietf-yang-types {
  prefix yang;
}
import ietf-inet-types {
  prefix inet;
}

organization "IETF";
contact
  "WG Web:    <http://tools.ietf.org/wg/netconf/>
  WG List:    <mailto:netconf@ietf.org>

  WG Chair: Mahesh Jethanandani
             <mailto:mjethanandani@gmail.com>

  WG Chair: Mehmet Ersue
             <mailto:mehmet.ersue@nokia.com>

  Editor: Eric Voit
           <mailto:evoit@cisco.com>

  Editor: Alexander Clemm
           <mailto:ludwig@clemm.org>

  Editor: Tim Jenkins
           <mailto:timjenki@cisco.com>

  Editor: Andy Bierman
           <mailto:andy@yumaworks.com>";

description
  "This module contains conceptual YANG specifications for
  notification messages with well known header objects.";

revision 2017-04-25 {
  description
    "This module includes definitions for two new yang
    notification message objects:
    (a) a message format including the definitions for common header
        information prior to notification payload.
    (b) a message format allowing the bundling of multiple
        notifications within it";

  reference
```

```
    "draft-voit-netconf-notification-messages-01";
}

/*
 * IDENTITIES
 */

/* Identities for notification record types */

identity notification-record-format {
  description
    "Base identity to represent a different formats for notification
    records.";
}

identity system-event {
  base notification-record-format;
  description
    "System XML event";
}

identity yang-datastore {
  base notification-record-format;
  description
    "yang data node / tree extract";
}

identity yang-patch {
  base notification-record-format;
  description
    "yang patch record";
}

identity syslog-entry {
  base notification-record-format;
  description
    "Unstructured syslog entry.";
}

identity alarm {
  base notification-record-format;
  description
    "Alarm (perhaps link draft-sharma-netmod-fault-model-01 for more
    info)";
}

/*
 * TYPEDEFS
 */
```

```
*/

typedef notification-record-format {
    type identityref {
        base notification-record-format;
    }
    description
        "Type of notification record";
}

/*
 * GROUPINGS
 */

grouping notification-message-header {
    description
        "Header information included with a notification.";
    leaf notification-id {
        type uint32;
        description
            "unique id for a notification which may go to one or many
            receivers.";
    }
    leaf notification-time {
        type yang:date-and-time;
        description
            "time the notification was generated prior to being sent to
            transport.";
    }
    leaf previous-notification-id {
        type uint32;
        description
            "Notification id previously sent by publisher to a specific
            receiver (allows detection of loss/duplication).";
    }
    leaf dscp {
        type inet:dscp;
        default "0";
        description
            "The push update's IP packet transport priority. This is made
            visible across network hops to receiver. The transport
            priority is shared for all receivers of a given
            subscription.";
    }
    leaf message-generator-id {
        type string;
        description
```

```
        "Software entity which created the notification message (e.g.,
        linecard 1).";
    }
    leaf signature {
        type string;
        description
            "Any originator signing of the contents of a notification
            message. This can be useful for originating applications to
            verify record contents even when shipping over unsecure
            transport.";
    }
}

grouping notification-record-header {
    description
        "Common informational objects which might help a receiver
        interpret the meaning, details, and importance of an event
        notification.";
    leaf record-time {
        type yang:date-and-time;
        mandatory true;
        description
            "Time the system recognized the occurrence of an event.";
    }
    leaf record-type {
        type notification-record-format;
        description
            "Describes the type of payload included. This is turn allow
            the interpretation of the record contents.";
    }
    leaf-list subscription-id {
        type uint32;
        description
            "Id of the subscription which led to the notification being
            generated.";
    }
    leaf record-id {
        type uint32;
        description
            "Identifier for the notification record.";
    }
    leaf observation-domain-id {
        type string;
        description
            "Software entity which created the notification record (e.g.,
            process id).";
    }
}
```

```
/*
 * NOTIFICATIONS
 */

notification notification-message {
  description
    "Notification message to a receiver containing only one event.";
  container notification-message-header {
    description
      "delineates header info from notification messages for easy
      parsing.";
    uses notification-record-header;
    uses notification-message-header;
  }
  anydata receiver-record-contents {
    description
      "Non-header info of what actually got sent to receiver after
      security filter. (Note: Possible to have extra process
      encryption.)";
  }
}

notification bundled-notification-message {
  description
    "Notification message to a receiver containing many events,
    possibly relating to independent subscriptions.";
  container bundled-notification-message-header {
    description
      "Delineates header info from notification messages for easy
      parsing.";
    uses notification-message-header {
      refine notification-time {
        mandatory true;
      }
    }
    leaf record-count {
      type uint16;
      description
        "Quantity of events in a bundled-notification-message
        for a specific receiver. This value is per receiver in
        case an entire notification record is filtered out.";
    }
  }
  list notification-records {
    description
      "Set of messages within a notification to a receiver.";
    container notification-record-header {
      description
```

```
        "delineates header info from notification messages for easy
        parsing.";
    uses notification-record-header;
}
anydata receiver-record-contents {
    description
        "Non-header info of what actually got sent to receiver after
        security filter. (Note: Possible to have extra process
        encryption.)";
}
}
}
}
}
<CODE ENDS>
```

7. Security Considerations

More needs to be thought through here, as this adds additional information onto notifications, the security implications shouldn't be significantly beyond those from [subscribe] other than ensuring that data plane devices can accomplish the necessary content filtering despite the potential of a new level of header being applied.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [subscribe] Voit, E., Clemm, A., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Notifications", April 2017, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-subscribed-notifications/>>.

8.2. Informative References

- [initial-version] Voit, E., Bierman, A., Clemm, A., and T. Jenkins, "Custom Subscription to Event Notifications", February 2017, <<https://tools.ietf.org/html/draft-voit-netmod-yang-notifications2-00>>.

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

[yang-push]

Clemm, A., Voit, E., Gonzalez Prieto, A., Prasad Tripathy, A., and E. Nilsen-Nygaard, "Subscribing to YANG datastore push updates", April 2017, <<https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>>.

Appendix A. Issues being worked

(To be removed by RFC editor prior to publication)

We need to define the ways to invoke and configure the capability within an originating device. This includes defining what header types are selected. This also includes knowing the header types which can be supported by a receiver.

We need to do a lot more to discuss transport efficiency implications.

Relationship with DTN protocols.

Appendix B. Querying an Object Model

(This appendix was in a previous iteration of this draft. It has been removed as unlikely to be seen in an implementation. It is being kept in for discussion purposes.)

It is also possible that that external entities might want to query message information after it has been sent. And therefore it is possible that an administrator would like to examine the contents of notifications via random access using a YANG model rather than Syslog. There could be several values in such random access. These include:

- o ability for applications to determine what message bundles were used to transport specific records.
- o ability for applications to check which receivers have been sent an event notification.
- o ability for applications to determine the time delta between event identification and transport.
- o ability to reconstruct message passing during troubleshooting.

- o ability to extract messages and records to evaluate whether the security filters have been properly applied.
- o ability to compare the payloads of the same notification message sent to different receivers (again to evaluate the impact of the security filtering).

If such random access is needed, it is possible to extend the YANG model data model document to enable random access to the information. A cut at what this might look like can be seen in [initial-version]

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Alexander Clemm
Huawei

Email: ludwig@clemm.org

Tim Jenkins
Cisco Systems

Email: timjenki@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2018

Z. Wang
G. Zheng
Huawei Technologies
July 3, 2017

Network Configuration Protocol (NETCONF) Proxy
draft-wangzheng-netconf-proxy-01

Abstract

This document presents Network Configuration Protocol (NETCONF) Proxy through which NETCONF requests can be forwarded to a target host. It would be useful when a client does not have direct network access to a target host.

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 January 4, 2018.

Copyright Notice

Copyright (c) 2017 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
1.1. Motivation	2
1.2. Netconf Proxy Use Case	4
1.2.1. Using Netconf Proxy to manage VNF Elements	4
1.2.2. Using NetConf Proxy to manage the Non-Gateway Elements of OSN (Optical Switch Network)	5
1.3. Requirements Terminology	6
2. Solution Overview	6
3. The NETCONF Client	8
4. The Proxy	10
5. The Target	10
6. New attribute: target-id	11
7. YANG DATA MODEL	12
7.1. Overview	12
7.2. YANG Module	12
8. Security Considerations	14
9. IANA Considerations	14
10. References	14
10.1. Normative References	14
10.2. Informative References	15
Authors' Addresses	15

1. Introduction

This document proposes a NETCONF Proxy mechanism. The mechanism extends the NETCONF protocol [RFC6241] which provides a standard way to set up NETCONF session. At its core, the mechanism defined here introduces a set of new operations which allow a client to forward NETCONF requests to a target host through an intermediary NETCONF proxy server, especially in case where client would otherwise not have direct network access to a target host. The document also includes YANG data model which extend the model and RPCs defined within [RFC6241].

1.1. Motivation

NETCONF provide a RPC-based mechanism to facilitate communication between a client and a server. The client can be a script or application typically running as part of a network manager. The server is typically a network device [RFC6241]. However, the network manager may not have direct network access to the target network devices. For example, some target network devices may locate in a network with private addresses behind a NAT device or a firewall. Thus, network manager cannot direct communicate with these target devices.

NETCONF Call Home [RFC8071] provides a mechanism that allows NETCONF Servers to initiate a connection with a NETCONF client, reversing the normal direction of NETCONF session setup. This allows a NETCONF Server, e.g. a networking device that needs to be managed, to reach out to a NETCONF Client, e.g. an Operations Support System of an SDN controller, in order to be managed. By reversing the direction in which NETCONF sessions are normally set up, problems such as establishing connectivity with devices behind a firewall can be alleviated. However, NETCONF Call Home requires that the server knows its client by way of configuration or discovery. It does not address the scenarios as presented below:

1. In some NFV scenarios, VNF instances are running in a private network. To reduce the management resources (like IP resources, bandwidth, etc) of large-scale management activities, these VNF instances may not be assigned IP addresses. Then the element management system (EMS), which located in public network, cannot be aware of the addresses of VNF instances. Therefore, the element management system (EMS) is difficult to manage these VNF instances via NETCONF protocol. More details please see section 1.2.1.
2. And in some cloud network scenarios, the gateway network element (GNE) and non-gateway network elements (N-GNEs) communicate with each other using some private protocol. And these non-gateway network elements (N-GNEs) may not IP devices. Therefore, the cloud centre EMS (element management system) cannot be aware of the addresses of N-GNEs. Thus, the element management system (EMS) is difficult to manage these N-GNE devices via NETCONF protocol. More details please see section 1.2.2.

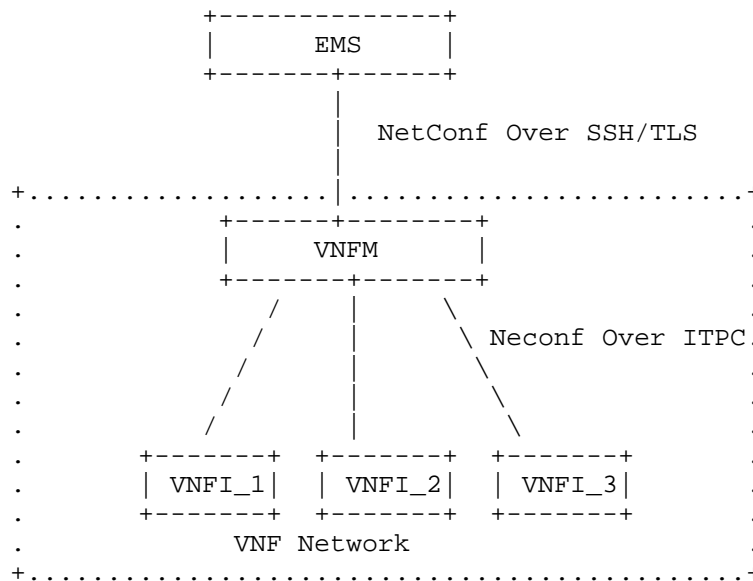
To solve that problem, this document proposes a NETCONF Proxy mechanism. The proxy can acts as an intermediary between manager and target device, therefore the client can set up a NETCONF session to a target through a NETCONF Proxy.

The mechanism allows the client to subsequently direct NETCONF requests to the server, to receive responses, and to subscribe to notifications from the server. While the NETCONF Proxy can be used to traverse NAT boundaries, it should be noted that it does not apply NAT mappings for contents carried as part of the NETCONF payload; specifically, it does not substitute IP address information that is carried as part of data nodes.

1.2. Netconf Proxy Use Case

1.2.1. Using Netconf Proxy to manage VNF Elements

Figure 1 illustrates EMS manage the VNF instances.



Using Netconf Proxy to manage VNF Elements

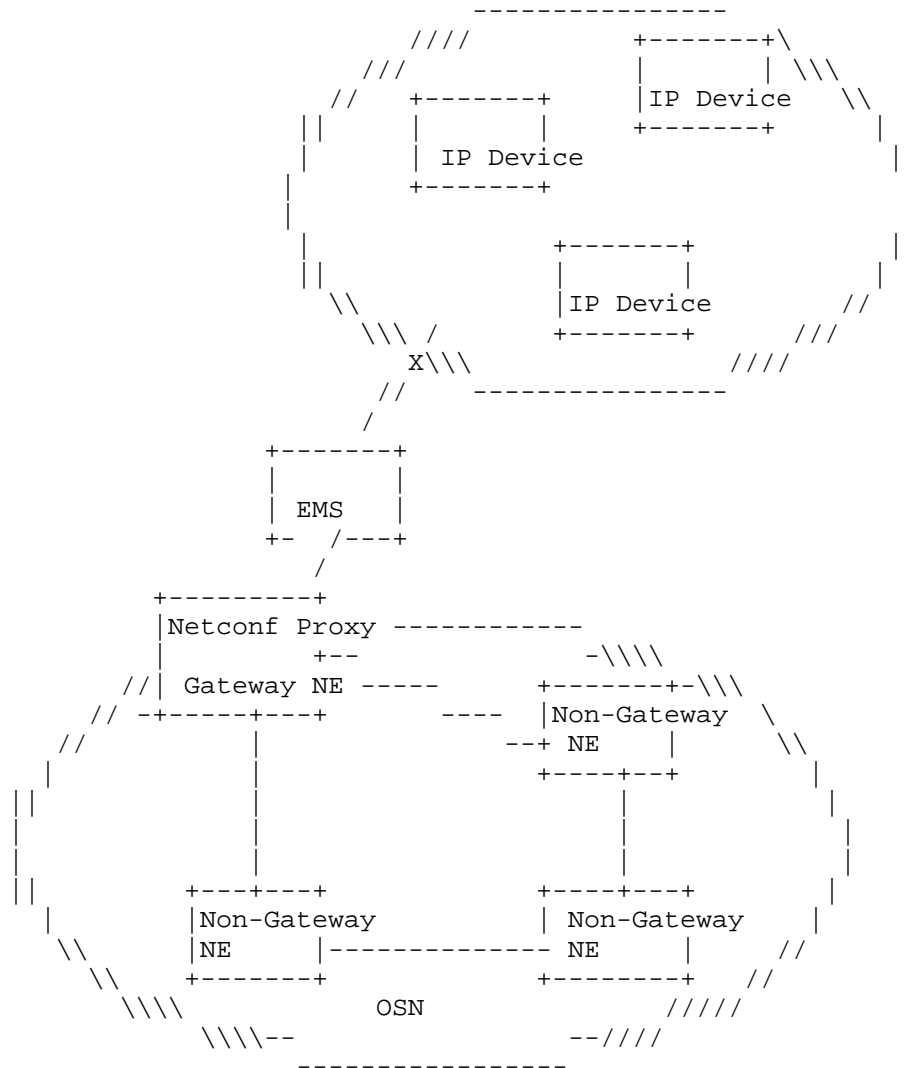
EMS is connected to VNFM through public network. To reduce the cost of management resources (like IP resources, bandwidth, etc) for large-scale management activities, the VNFIs(VNF instances) are running a TIPC(Transparent Inter-process Communication) protocol, and these VNIs are not assigned IP address. The management data of VNFIs will be transported to VNFM via TIPC. Within the VNF Network, the TIPC protocol will provide the data to the respective application i.e. NETCONF.

To manage the VNFIs, EMS will access the VNFIs via the Netconf Proxy which located in the VNFM. EMS is access to Netconf Proxy through Netconf over SSH. Within the VNF network, the NETCONF data will be transported from VNFM to VNFIs over Transparent Inter-process Communication (TIPC) protocol. And the VNFIs will report their IDs and other information to netconf proxy. The netconf proxy will store these information in the "target-list". According to these

information, the EMS can manage the VNFIs via Netconf Proxy, more details see section 2.

1.2.2. Using NetConf Proxy to manage the Non-Gateway Elements of OSN (Optical Switch Network)

Figure 2 illustrates EMS manage the Non-Gateway Elements of Optical Switch Network (OSN) via Netconf Proxy.



Using Netconf Proxy to manage VNF Elements

The network between EMS and GNE is IP Accessible whereas the network between GNE and N-GNE is not IP Accessible. Therefore, the EMS cannot be aware of the address of N-GENs. Note that the Non-Gateway Elements are not IP devices, thus the N-GNE cannot support NAT. The management data of N-GNE will be transported to GNE on OSN's private transmission layer. Within the OSN Network Elements, the OSN private transmission protocol (i.e. via QX interface [G.773]) will provide the data to the respective application i.e. NETCONF.

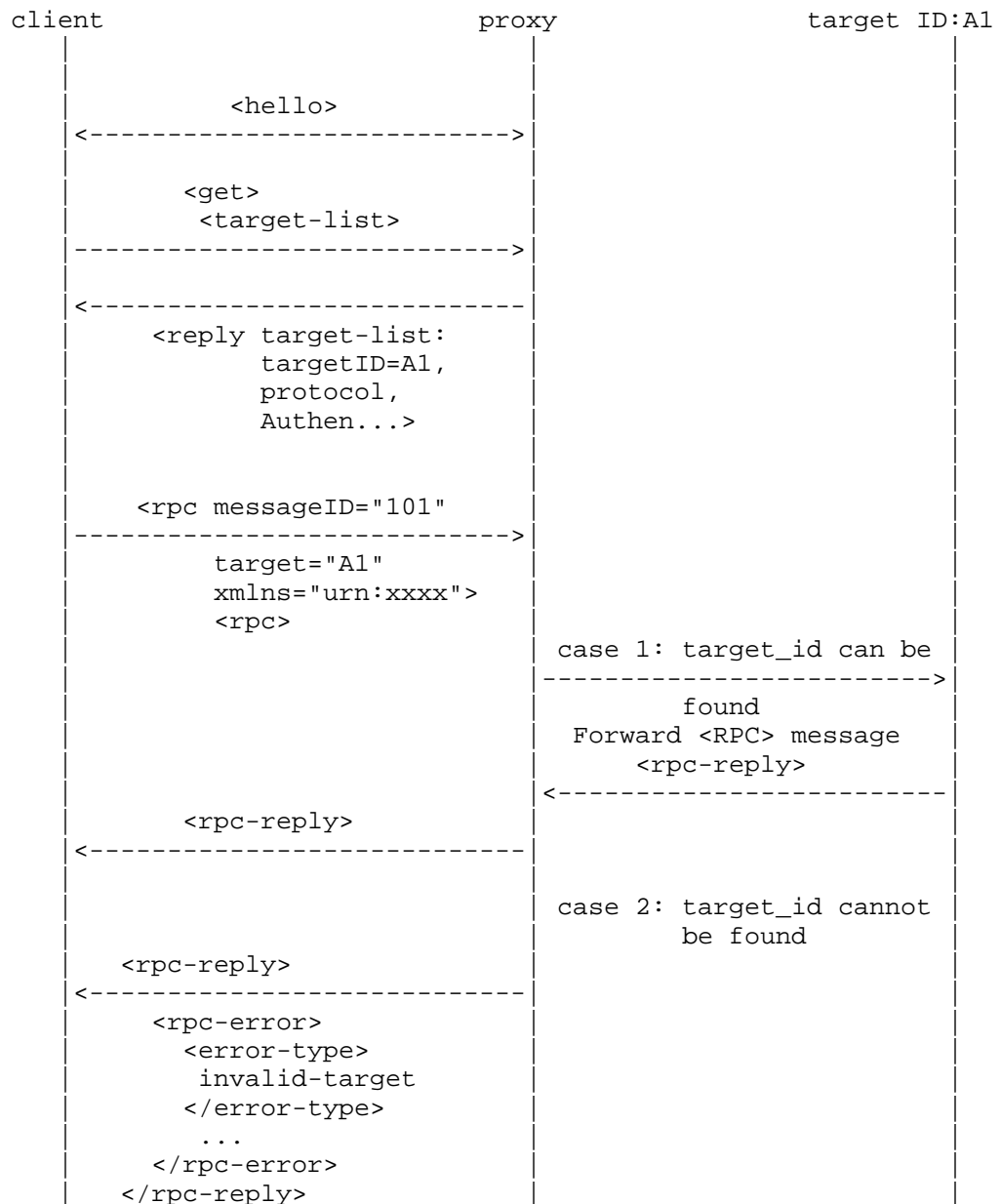
To manage the non-gateway network elements, NMS will access the non-gateway NE via the Netconf Proxy which located in the gateway network element (GNE). EMS is access to Netconf Proxy through Netconf over SSH. Within the OSN, the NETCONF data will be transported from GNE to Non-GNEs over OSN private transport protocol. And the Non-GNEs will report their IDs and other information to netconf proxy. The netconf proxy will store these information in the "target-list". According to these information, the EMS can manage the Non-Gateway Elements of OSN via Netconf Proxy, more details see section 2.

1.3. Requirements 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].

2. Solution Overview

The diagram below illustrates how the client can set up a NETCONF session to a target through the NETCONF proxy.



This diagram makes the following points:

1. The client initiates the connection using the SSH/TLS transport protocol. When the NETCONF session is established, the client

and proxy MUST send a <hello> element containing a list of that peer's capabilities. The proxy SHOULD send at least the "netconf" and "proxy" capabilities. And other rules of capabilities exchange described in section 8 of [RFC6241].

2. The client sends a <get> RPC to proxy to retrieve the "target-list" of the proxy.
3. The proxy responds with a <get-reply> RPC which containing "target-list" attributes. The "target-list" attributes includes the target's information such as target-id, protocol, authentication, etc.
4. The client receives a the <get-reply> RPC from the proxy, and retrieves the target information according to the received "target-list". Subsequently, the client can direct NETCONF requests to the target according to the received "target-list", to receive responses, and to subscribe to notifications from the target. For example, the client wants to retrieve the configuration information of a target. The client should construct a <get-config> message according to the received "target-list". This <get-config> message SHOULD contain at least a "target-id" attribute. And then client sends this <get-config> message to proxy and waits for a reply.
5. The proxy receives the RPC message and checks the value of "target-id" attribute:
 - If the target is not found, then an "invalid-target" error will be returned.
 - If the target can be found, then proxy forwards the RPC message, which received from client, to corresponding target.
6. The target receives the RPC message. And then sends an <rpc-reply> message in response to the received RPC message.

3. The NETCONF Client

The term "client" is defined in [RFC6241], Section 1.1 "client". In the context of network management, the NETCONF client might be a network management system for example a EMS (element management system).

The client operation describes as follows:

1. The client initiates a connection to proxy using the SSH/TLS transport protocol [RFC6242]. How to establish an SSH/TLS transport connection is described in [RFC6242]

2. When the NETCONF session is established, the client sends a <hello> message to proxy, then waits for a reply. This <hello> message contains a list of client's capabilities.

3. After capabilities exchange, the client sends a <get> RPC to proxy to retrieve the "target-list" of the proxy, then waits for a reply.

4. The client receives the <get-reply> RPC from the proxy, looks up the value of "target-list", and then constructs a RPC message according to the received "target-list".

For example, the client wants to retrieve the configuration information of a target A1. The client should construct a <get-config> message. This <get-config> message SHOULD contain at least a "target-id" attribute:

```
<rpc message-id="101"
  target-id="A1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.2/config">
        <users/>
      </top>
    </filter>
  </get-config>
</rpc>
```

And then client sends this <hello> message to proxy and waits for a reply.

5. If the reply containing a <data> element which satisfied with "Positive Response" condition of corresponding RPC (Section 7 of [RFC6241]), it means that the client has successfully managed the target device.

6. If the reply contains the "invalid-target" error, the process turn to step (4) or aborts.

7. Otherwise, the client interprets the error and aborts.

4. The Proxy

The Proxy should ensure that requests given by client for a particular target device should reach the target device and the operations should be executed on that target device and the response should be given back to the client.

The proxy operation describes as follows:

1. When the NETCONF session is established, the proxy sends a <hello> element containing a list of proxy's capabilities. The proxy SHOULD send at least the "netconf" and "proxy" capabilities. And other rules of capabilities exchange described in section 8 of [RFC6242].

2. The proxy receives the <get> RPC and then responds with a <get-reply> RPC which containing "target-list" attributes. The "target-list" attributes SHOULD includes the target's information such as target-id, protocol, etc. The following example shows a "target-list":

```
<target-list>
  <target-id>A1</target-id>
  <protocol>protocol-foo</protocol>
</target-list>
```

3. The proxy receives the RPC message and checks the value of "target-id" attribute:

If the target is not found in target-list, then an "invalid-target" error will be returned.

If the target can be found, then proxy forwards the RPC message, which received from client, to corresponding target.

4. In this Netconf-Proxy model, the proxy reads data from both the client and the target, and writes any data received to the other end, without interpreting the data. If any side of the connection is closed, the proxy closes the other side.

5. The Target

The term "target" is equivalent to the term "server" which is defined in [RFC6242], Section 1.1 "server". In the context of network management, the target is typically a network device.

The target operations describes as follows:

If the connection between the proxy and the target established. And target receives the RPC message from the proxy, and then responds a <rpc-reply> message.

If the target can satisfy the RPC request, the target sends an <rpc-reply> element containing a <data> element which satisfied with "Positive Response" condition of corresponding RPC (Section 7 of [RFC6241]).

If an error occurs during the processing of an <rpc> request, the target sends an <rpc-reply> element which includes a corresponding <rpc-error> element (Section 7 of [RFC6241]).

6. New attribute: target-id

A proxy can be used by a client to connect to several servers and to maintain multiple NETCONF sessions. A client may use the proxy even to maintain multiple NETCONF sessions with the same NETCONF server. When issuing a NETCONF request, a client must therefore differentiate between NETCONF sessions. To solve this problem, a new attribute "target-id" is defined. This attribute allow the proxy to forward RPC to corresponding target.

For example:

The following <rpc> element invokes the NETCONF <get> method and includes the "target-id" attribute:

```
<rpc message-id="101"
  target-id="A1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get/>
</rpc>

<rpc-reply message-id="101"
  target-id="A1"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <!-- contents here... -->
  </data>
</rpc-reply>
```

7. YANG DATA MODEL

7.1. Overview

The YANG data model for NETCONF Proxy is depicted in the following figure. Following Yang tree convention in the depiction, brackets enclose list keys, "rw" means configuration, "ro" operational state data, "?" designates optional nodes, "*" designates nodes that can have multiple instances. A "+" at the end of a line indicates that the line is to be concatenated with the subsequent line.

```
module: ietf-netconf-proxy
  +--rw proxy {proxy}?
    +--rw proxy-name?      string
    +--rw target-list* [target-id]
      +--rw target-id      string
      +--rw protocol?      string
      +--rw authentication? string
```

7.2. YANG Module

```
<CODE BEGINS> file "ietf-netconf-proxy@2017-03-09.yang"
module ietf-netconf-proxy {

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-proxy";

  prefix np;

  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "WG Web:  http://tools.ietf.org/wg/netconf
    WG List:  netconf@ietf.org

    WG Chair: Mehmet Ersue
              mehmet.ersue@nsn.com

    Editor:   zitao wang
              wangzitao@huawei.com";

  description
    "NETCONF Protocol Data Types and Protocol Operations.

    Copyright (c) 2011 IETF Trust and the persons identified as
    the document authors. All rights reserved."
```

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This YANG module describe how to define a netconf proxy";

```
revision 2017-03-09 {
  description
    "Initial revision";
  reference
    "draft-wang-netconf-proxy";
}

feature proxy {
  description
    "Netconf proxy";
}

container proxy {
  if-feature proxy;
  leaf proxy-name{
    type string;
    description
      "Proxy name";
  }
  list target-list {
    key "target-id";
    leaf target-id{
      type string;
      description
        "Target ID";
    }
    leaf protocol {
      type string;
      description
        "Support protocols";
    }
    leaf authentication {
      type string;
      description
        "Authentication";
    }
    description
      "List for target information";
  }
}
```

```
    description
    "Container for NETCONF Proxy";
  }
}
<CODE ENDS>
```

8. Security Considerations

The security considerations described in [RFC6242] and [RFC7589], and by extension [RFC4253], [RFC5246] apply here as well.

9. IANA Considerations

TBD

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<http://www.rfc-editor.org/info/rfc4253>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [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>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <<http://www.rfc-editor.org/info/rfc7589>>.

- [RFC793] Postel, J., "TRANSMISSION CONTROL PROTOCOL", STD 7, September 1981, <<https://www.ietf.org/rfc/rfc793.txt>>.

10.2. Informative References

- [G.773] "Protocol suites for Q-interfaces for management of transmission systems", ITU-T Recommendation G.773, 1993.

Authors' Addresses

Zitao Wang
Huawei Technologies
101 Software Avenue, Yuhua District
Nanjing
China

EMail: wangzitao@huawei.com

Guangying Zheng
Huawei Technologies
101 Software Avenue, Yuhua District
Nanjing
China

EMail: zhengguangying@huawei.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2018

G. Zheng
T. Zhou
A. Clemm
Huawei
July 2, 2017

UDP based Publication Channel for Streaming Telemetry
draft-zheng-netconf-udp-pub-channel-00

Abstract

This document describes a UDP-based publication channel for streaming telemetry use to collect data from devices. A new shim header is proposed to facilitate the distributed data collection mechanism which directly pushes data from line cards to the collector. Because of the lightweight UDP encapsulation, higher frequency and better transit performance can be achieved.

Requirements Language

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].

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 January 3, 2018.

Copyright Notice

Copyright (c) 2017 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	4
3. Solution Overview	4
4. UDP Transport for Publication Channel	6
4.1. Data Format	6
4.2. Options	8
4.2.1. Reliability Option	8
4.2.2. Authentication Option	9
4.2.3. Encryption Option	9
4.3. Data encoding	9
5. YANG Data Model for Subscription Management	9
6. Retransmission Request	10
7. IANA Considerations	10
8. Security Considerations	10
9. Acknowledgements	10
10. References	10
10.1. Normative References	10
10.2. Informative References	11
Appendix A. An Appendix	11
Authors' Addresses	11

1. Introduction

Streaming telemetry refers to sending a continuous stream of operational data from a device to a remote receiver. This provides an ability to monitor a network from remote and to provide network analytics. Devices generate telemetry data and push that data to a collector for further analysis. By streaming the data, much better performance, finer-grained sampling, monitoring accuracy, and bandwidth utilization can be achieved than with polling-based alternatives.

Sub-Notif [I-D.ietf-netconf-subscribed-notifications] and YANG-Push [I-D.ietf-netconf-yang-push] defines a mechanism that allows a collector to subscribe to updates of YANG-defined data that is maintained in a YANG [RFC7950] datastore. The mechanism separates

the management and control of subscriptions from the transport that is used to actually stream and deliver the data. Two transports have been defined so far, Netconf and Restconf/HTTP2.

While powerful in its features and general in its architecture, in its current form the mechanism needs to be extended to stream telemetry data at high velocity from devices that feature a distributed architecture. Specifically, there are two aspects that need to be addressed:

1. The transports that have been defined so far, Netconf and HTTP2, are ultimately based on TCP and lack the efficiency needed to stream data continuously at high velocity. A lightweight, more efficient transport, e.g. a transport based on UDP is needed.
 - * Firstly, data collector will suffer a lot of TCP connections from many line cards equipped on different devices.
 - * Secondly, as no connection state needs to be maintained, UDP encapsulation can be easily implemented by hardware which will further improve the performance.
 - * Thirdly, because of the lightweight UDP encapsulation, higher frequency and better transit performance can be achieved, which is important for streaming telemetry.
2. The current design involves a single push server. In the case of data originating from multiple line cards, the design requires data to be internally forwarded from those line cards to the push server, presumably on a main board, which then combines the individual data items into a single consolidated stream. This centralized data collection mechanism can result in a performance bottleneck, especially when large amounts of data are involved. What is needed instead is support for a distributed mechanism that allows to directly push multiple individual substreams, e.g. one from each line card, without needing to first pass them through an additional processing stage for internal consolidation, but still allowing those substreams to be managed and controlled via a single subscription.

This document specifies a distributed data collection mechanism which can directly push data from line cards to a collector by using a UDP based publication channel. Specifically, the following are specified:

- o A higher-performance transport option for YANG-Push that leverages UDP.

- o Extensions to YANG-Push's subscription model that allow a single subscription to control multiple internal data originators that each generate their own independent telemetry streams. Note: Because the ability to support multiple streams via a single subscription might be applicable to other transports as well, this aspect might be split into a separate specification in future revisions of this draft.

While this document will focus on the data publication channel, the subscription can be used in conjunction with the mechanism proposed in [I-D.ietf-netconf-yang-push] with necessary extensions.

2. Terminology

Streaming telemetry: refers to sending a continuous stream of operational data from a device to a remote receiver. This provides an ability to monitor a network from remote and to provide network analytics.

Component subscription: A subscription that defines the data from each individual entity which is managed and controlled by a single subscription server.

Subscription agent: An agent that streams telemetry data per the terms of a component subscription.

3. Solution Overview

The typical distributed data collection solution is shown in figure 1. The subscription server located in the main board receives the subscription requests or configurations. It may be colocated, not necessary, with a Netconf server which interacts with outside clients. When receiving a subscription request, the subscription server decomposes the subscription into multiple component subscriptions, each involving data from a separate internal telemetry source, for example a line card. The component subscriptions are distributed within the device to the subscription agents located in line cards. Subsequently, each line card generates its own stream of telemetry data, collecting and encapsulating the packets per the component subscription and streaming it to the designated data collector.

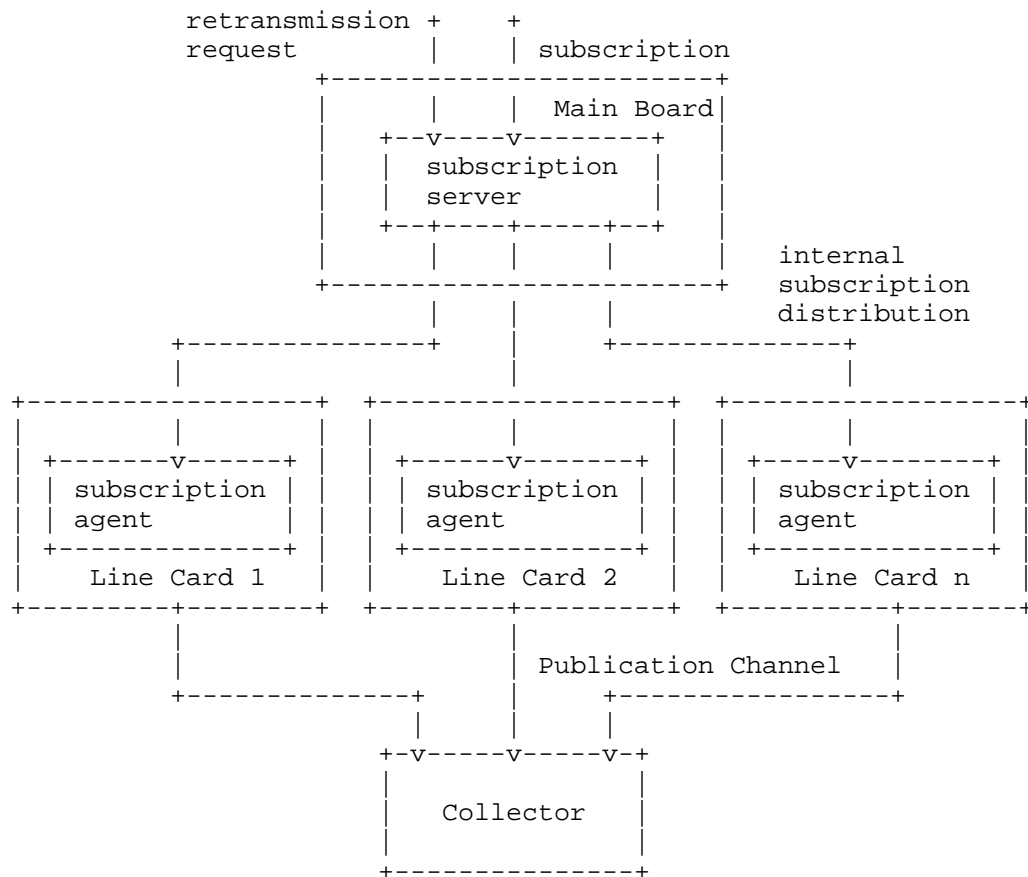
The publication channel supports the reliable data streaming, for example for some alarm events. The subscriber has the option of deducing the packet loss and the disorder based on the information carried by the notification data. And the subscriber will decide the behavior to request retransmission. The subscriber can send the

retransmission request to the subscriber server for further processing.

Subscription server and subscription agents interact with each other in several ways:

- o Subscription agents need to have a registration or announcement handshake with the subscription server, so the subscription server is aware of them and of lifecycle events (such as subscription agents appearing and disappearing).
- o The subscription server relays the component subscriptions to the subscription agents.
- o The subscription agents indicate status of component subscriptions to the subscription server. The status of the overall "master" subscription is maintained by the subscription server. The subscription server is also responsible for notifying the subscriber in case of any problems of component subscriptions.

The rest of the draft describes the UDP based publication channel.



4. UDP Transport for Publication Channel

In [I-D.voit-netconf-notification-messages], the transport independent message header is proposed for the notification use. The following shim header refers to and implements that message header definition.

4.1. Data Format

The data format of the UDP based based publication transport is shown as follows.

- o The Timestamp, including the second part and the microsecond part, indicate the time the message was packaged and sent to the receiver. The Timestamp is defined per RFC 3339.
- o The details of the Options will be described in the respective sections.

After the inform header is the real content which is encoded. The actual encoding is based on the subscription, e.g., in binary with GPB or CBOR [RFC7049].

More details of the content encoding is TBD.

4.2. Options

4.2.1. Reliability Option

The UDP based publication transport described in this document provides two streaming modes, the reliable mode and the unreliable mode, for different SLA (Service Level Agreement) and telemetry requirements.

In the unreliable streaming mode, the line card pushes the encapsulated data to the data collector without any sequence information. So the subscriber does not know whether the data is correctly received or not. Hence no retransmission happens.

The reliable streaming mode provides sequence information in the UDP packet, based on which the subscriber can deduce the packet loss and disorder. Then the subscriber can decide whether to request the retransmission of the lost packets.

In most case, the unreliable streaming mode is preferred. Because the reliable streaming mode will cost more network bandwidth and precious device resource. Different from the unreliable streaming mode, the line card cannot remove the sent reliable notifications immediately, but to keep them in the memory for a while. Reliable notifications may be pushed multiple times, which will increase the traffic. When choosing the reliable streaming mode or the unreliable streaming mode, the operator needs to consider the reliable requirement together with the resource usage.

When the reliability flag is set to 1. The following option will be attached


```
+--rw subscription-config {configured-subscriptions}?  
| + ...  
| +--rw receivers  
| | + ...  
| | +--rw protocol? transport-protocol  
| | | +--rw udp-transport-type? udp-transport-type  
| | | +--rw reliable?  
| | | +--rw authentication?  
| | | +--rw encryption?
```

As in the above YANG tree, when the transport protocol is set to UDP, retries indicates the maximum retry times of the reliable streaming mode, and the timeout indicates the time out for retry in reliable streaming mode.

TBD. Note this YANG tree just to show we need to extend subscription mode, including the configurations and the RPCs. More details will be added later.

6. Retransmission Request

TBD

7. IANA Considerations

TBD

8. Security Considerations

9. Acknowledgements

The authors of this documents would like to thank Eric Voit, Tim Jenkins, and Huiyang Yang for the initial comments.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.

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

10.2. Informative References

- [I-D.ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Custom Subscription to Event Notifications", draft-ietf-netconf-subscribed-notifications-02 (work in progress), April 2017.
- [I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-07 (work in progress), June 2017.
- [I-D.voit-netconf-notification-messages]
Voit, E., Bierman, A., Clemm, A., and T. Jenkins, "Notification Message Headers and Bundles", draft-voit-netconf-notification-messages-00 (work in progress), April 2017.

Appendix A. An Appendix

Authors' Addresses

Guangying Zheng
Huawei
Nanjing, Jiangsu
China

Fax: +
Email: zhengguangying@huawei.com

Tianran Zhou
Huawei
156 Beiqing Rd., Haidian District
Beijing
China

Email: zhoutianran@huawei.com

Alexander Clemm
Huawei
2330 Central Expressway
Santa Clara, California
USA

Email: alexander.clemm@huawei.com