NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 6, 2018

K. Watsen
Juniper Networks

June 4, 2018

Common YANG Data Types for Cryptography
draft-ietf-netconf-crypto-types-00

Abstract

This document defines YANG identities and typedefs useful for cryptographic applications.

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 "2018-06-04" --> 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 https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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. YANG Module 3
3. Security Considerations 11
4. IANA Considerations 11
  4.1. The IETF XML Registry 11
  4.2. The YANG Module Names Registry 12
5. References 12
  5.1. Normative References 12
  5.2. Informative References 13
Appendix A. Change Log 14
  A.1. I-D to 00 14
Acknowledgements 14
Author's Address 14

1. Introduction

This document defines a YANG 1.1 [RFC7950] module specifying identities and typedefs useful for cryptography.

As the YANG module only defines identities and typedefs, this draft does not present a YANG tree diagram [RFC8340] or any examples illustrating usage of the module.

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.

2. YANG Module

This module has normartive references for [RFC4253], [RFC5280], [RFC5480], [RFC5652], and [ITU.X690.2015] and has informational references to [RFC6234] and [RFC8017]

```
<CODE BEGINS> file "ietf-crypto-types@2018-06-04.yang"
module ietf-crypto-types {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-crypto-types";
  prefix "ct";

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

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

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

  description
   "This module defines common YANG types for cryptographic
    applications.

    Copyright (c) 2018 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 "2018-06-04" {
    description
     "Initial version";
    reference
     "RFC XXXX: Common YANG Data Types for Cryptography";
```

```
     }


     /****************************************/
     /*   Identities for Hashing Algorithms   */
     /****************************************/

     identity hash-algorithm {
       description
         "A base identity for hash algorithm verification.";
     }

     identity sha-256 {
       base "hash-algorithm";
       description "The SHA-256 algorithm.";
       reference "RFC 6234: US Secure Hash Algorithms.";
     }

     /*************************************/
     /*   Identities for Key Algorithms   */
     /*************************************/

     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
         "RFC 8017:
            PKCS #1: RSA Cryptography Specifications Version 2.2.";
     }

     identity rsa2048 {
       base key-algorithm;
       description
         "The RSA algorithm using a 2048-bit key.";
       reference
         "RFC 8017:
            PKCS #1: RSA Cryptography Specifications Version 2.2.";
     }

     identity rsa3072 {
       base key-algorithm;
       description
```

```
        "The RSA algorithm using a 3072-bit key.";
      reference
        "RFC 8017:
           PKCS #1: RSA Cryptography Specifications Version 2.2.";
    }

    identity rsa4096 {
      base key-algorithm;
      description
        "The RSA algorithm using a 4096-bit key.";
      reference
        "RFC 8017:
           PKCS #1: RSA Cryptography Specifications Version 2.2.";
    }

    identity rsa7680 {
      base key-algorithm;
      description
        "The RSA algorithm using a 7680-bit key.";
      reference
        "RFC 8017:
           PKCS #1: RSA Cryptography Specifications Version 2.2.";
    }

    identity rsa15360 {
      base key-algorithm;
      description
        "The RSA algorithm using a 15360-bit key.";
      reference
        "RFC 8017:
           PKCS #1: RSA Cryptography Specifications Version 2.2.";
    }

    identity secp192r1 {
      base key-algorithm;
      description
        "The secp192r1 algorithm.";
      reference
        "RFC 5480: Elliptic Curve Cryptography Subject Public
                   Key Information.";
    }

    identity secp256r1 {
      base key-algorithm;
      description
        "The secp256r1 algorithm.";
      reference
        "RFC 5480: Elliptic Curve Cryptography Subject Public
```

```
                    Key Information.";
     }

     identity secp384r1 {
       base key-algorithm;
       description
         "The secp384r1 algorithm.";
       reference
         "RFC 5480: Elliptic Curve Cryptography Subject Public
                    Key Information.";
     }

     identity secp521r1 {
       base key-algorithm;
       description
         "The secp521r1 algorithm.";
       reference
         "RFC 5480: Elliptic Curve Cryptography Subject Public
                    Key Information.";
     }


     /*********************************************************/
     /*   Typedefs for identityrefs to above base identites   */
     /*********************************************************/

     typedef hash-algorithm-ref {
       type identityref {
         base "hash-algorithm";
       }
       description
         "This typedef enables importing modules to easily define an
          identityref to the 'hash-algorithm' base identity.";
     }

     typedef key-algorithm-ref {
       type identityref {
         base "key-algorithm";
       }
       description
         "This typedef enables importing modules to easily define an
          identityref to the 'key-algorithm' base identity.";
     }


     /****************************************************/
     /*   Typedefs for ASN.1 structures from RFC 5280   */
     /****************************************************/
```

```
typedef x509 {
  type binary;
  description
    "A Certificate structure, as specified in RFC 5280,
     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).";
}

typedef crl {
  type binary;
  description
    "A CertificateList structure, as specified in RFC 5280,
     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).";
}

/***********************************************/
/*   Typedefs for ASN.1 structures from 5652   */
/***********************************************/

typedef cms {
  type binary;
  description
    "A ContentInfo structure, as specified in RFC 5652,
     encoded using ASN.1 distinguished encoding rules (DER),
     as specified in ITU-T X.690.";
  reference
    "RFC 5652:
       Cryptographic Message Syntax (CMS)
     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).";
     }

     typedef data-content-cms {
       type cms;
       description
         "A CMS structure whose top-most content type MUST be the
          data content type, as described by Section 4 in RFC 5652.";
       reference
         "RFC 5652: Cryptographic Message Syntax (CMS)";
     }

     typedef signed-data-cms {
       type cms;
       description
         "A CMS structure whose top-most content type MUST be the
          signed-data content type, as described by Section 5 in
          RFC 5652.";
       reference
         "RFC 5652: Cryptographic Message Syntax (CMS)";
     }

     typedef enveloped-data-cms {
       type cms;
       description
         "A CMS structure whose top-most content type MUST be the
          enveloped-data content type, as described by Section 6
          in RFC 5652.";
       reference
         "RFC 5652: Cryptographic Message Syntax (CMS)";
     }

     typedef digested-data-cms {
       type cms;
       description
         "A CMS structure whose top-most content type MUST be the
          digested-data content type, as described by Section 7
          in RFC 5652.";
       reference
         "RFC 5652: Cryptographic Message Syntax (CMS)";
     }

     typedef encrypted-data-cms {
       type cms;
       description
```

```
         "A CMS structure whose top-most content type MUST be the
          encrypted-data content type, as described by Section 8
          in RFC 5652.";
       reference
         "RFC 5652: Cryptographic Message Syntax (CMS)";
     }

     typedef authenticated-data-cms {
       type cms;
       description
         "A CMS structure whose top-most content type MUST be the
          authenticated-data content type, as described by Section 9
          in RFC 5652.";
       reference
         "RFC 5652: Cryptographic Message Syntax (CMS)";
     }

     /****************************************************/
     /*   Typedefs for structures related to RFC 4253   */
     /****************************************************/

     typedef ssh-host-key {
       type binary;
       description
         "The binary public key data for this SSH key, as
          specified by RFC 4253, Section 6.6, i.e.:

            string    certificate or public key format
                      identifier
            byte[n]   key/certificate data.";
       reference
         "RFC 4253: The Secure Shell (SSH) Transport Layer
                    Protocol";
     }

     /*********************************************************/
     /*   Typedefs for ASN.1 structures related to RFC 5280   */
     /*********************************************************/

     typedef trust-anchor-cert-x509 {
       type x509;
       description
         "A Certificate structure that MUST encode a self-signed
          root certificate.";
     }

     typedef end-entity-cert-x509 {
       type x509;
```

```
      description
        "A Certificate structure that MUST encode a certificate
         that is neither self-signed nor having Basic constraint
         CA true.";
    }

    /*********************************************************/
    /*   Typedefs for ASN.1 structures related to RFC 5652   */
    /*********************************************************/

    typedef trust-anchor-cert-cms {
      type signed-data-cms;
      description
        "A CMS SignedData structure that MUST contain the chain of
         X.509 certificates needed to authenticate the certificate
         presented by a client or end-entity.

         The CMS MUST contain only a single chain of certificates.
         The client or end-entity certificate MUST only authenticate
         to last intermediate CA certificate listed in the chain.

         In all cases, the chain MUST include a self-signed root
         certificate.  In the case where the root certificate is
         itself the issuer of the client or end-entity certificate,
         only one certificate is present.

         This CMS structure MAY (as applicable where this type is
         used) also contain suitably fresh (as defined by local
         policy) revocation objects with which the device can
         verify the revocation status of the certificates.

         This CMS encodes the degenerate form of the SignedData
         structure that is commonly used to disseminate X.509
         certificates and revocation objects (RFC 5280).";
      reference
        "RFC 5280:
           Internet X.509 Public Key Infrastructure Certificate
           and Certificate Revocation List (CRL) Profile.";
    }

    typedef end-entity-cert-cms {
      type signed-data-cms;
      description
        "A CMS SignedData structure that MUST contain the end
         entity certificate itself, and MAY contain any number
         of intermediate certificates leading up to a trust
         anchor certificate.  The trust anchor certificate
         MAY be included as well.
```

```
          The CMS MUST contain a single end entity certificate.
          The CMS MUST NOT contain any spurious certificates.

          This CMS structure MAY (as applicable where this type is
          used) also contain suitably fresh (as defined by local
          policy) revocation objects with which the device can
          verify the revocation status of the certificates.

          This CMS encodes the degenerate form of the SignedData
          structure that is commonly used to disseminate X.509
          certificates and revocation objects (RFC 5280).";
       reference
         "RFC 5280:
           Internet X.509 Public Key Infrastructure Certificate
           and Certificate Revocation List (CRL) Profile.";
     }

   }
   <CODE ENDS>
```

3. Security Considerations

In order to use YANG identities for algorithm identifiers, only the most commonly used RSA key lengths are supported for the RSA algorithm. Additional key lengths can be defined in another module or added into a future version of this document.

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

The YANG module defined in this document specifies only typedefs and identities, and hence there are no YANG-specific security considerations that need to be addressed.

4. IANA Considerations

4.1. The IETF XML Registry

This document registers one URI in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

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

4.2. 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-crypto-types
namespace:    urn:ietf:params:xml:ns:yang:ietf-crypto-types
prefix:       ct
reference:    RFC XXXX
```

5. References

5.1. Normative References

[ITU.X690.2015]
International Telecommunication Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, August 2015, <https://www.itu.int/rec/T-REC-X.690/>.

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

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <https://www.rfc-editor.org/info/rfc4253>.

[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, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

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

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

5.2. Informative References

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://www.rfc-editor.org/info/rfc6020>.

[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, <https://www.rfc-editor.org/info/rfc6234>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Change Log

A.1. I-D to 00

- o Removed groupings and notifications.
- o Added typedefs for identityrefs.
- o Added typedefs for other RFC 5280 structures.
- o Added typedefs for other RFC 5652 structures.
- o Added convenience typedefs for RFC 4253, RFC 5280, and RFC 5652.

Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Martin Bjorklund, Balazs Kovacs, Eric Voit, and Liang Xia.

Author’s Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 6, 2018

K. Watsen
Juniper Networks

June 4, 2018

YANG Data Model for a Centralized Keystore Mechanism
draft-ietf-netconf-keystore-05

Abstract

This document defines a YANG 1.1 module called "ietf-keystore" that enables centralized configuration of asymmetric keys and their associated certificates, and notification for when configured certificates are about to expire.

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:

- "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:

- "2018-06-04" --> the publication date of this draft

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

- 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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. Requirements Language 3
3. The Keystore Model 4
  3.1. Tree Diagram 4
  3.2. Example Usage 6
  3.3. YANG Module 12
4. Security Considerations 21
5. IANA Considerations 23
  5.1. The IETF XML Registry 23
  5.2. The YANG Module Names Registry 23
6. References 23
  6.1. Normative References 23
  6.2. Informative References 24
Appendix A. Change Log 26
  A.1. 00 to 01 26
  A.2. 01 to 02 26
  A.3. 02 to 03 26
  A.4. 03 to 04 26
  A.5. 04 to 05 27
Acknowledgements 27
Author's Address 27

1. Introduction

This document defines a YANG 1.1 [RFC7950] module called "ietf-keystore" that enables centralized configuration of asymmetric keys and their associated certificates, and notification for when configured certificates are about to expire.

This module also defines Six groupings designed for maximum reuse. These groupings include one for the public half of an asymmetric key, one for both the public and private halves of an asymmetric key, one for both halves of an asymmetric key and a list of associated certificates, one for an asymmetric key that may be configured locally or via a reference to an asymmetric key in the keystore, one for a trust anchor certificate and, lastly, one for an end entity certificate.

Special consideration has been given for systems that have cryptographic hardware, such as a Trusted Protection Module (TPM). These systems are unique in that the cryptographic hardware completely hides the private keys and must perform all private key operations. To support such hardware, the "private-key" can be the special value "hardware-protected" and the actions "generate-private-key" and "generate-certificate-signing-request" can be used to direct these operations to the hardware .

This document in compliant with Network Management Datastore Architecture (NMDA) [RFC8342]. For instance, to support keys and associated certificates installed during manufacturing (e.g., for a IDevID [Std-802.1AR-2009] certificate), it is expected that such data may appear only in <operational>.

While only asymmetric keys are currently supported, the module has been designed to enable other key types to be introduced in the future.

The module does not support protecting the contents of the keystore (e.g., via encryption), though it could be extended to do so in the future.

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

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. The Keystore Model

3.1. Tree Diagram

This section provides a tree diagrams [RFC8340] for the "ietf-keystore" module that presents both the protocol-accessible "keystore" as well the all the groupings intended for external usage.

```
module: ietf-keystore
  +--rw keystore
     +--rw asymmetric-keys
        +--rw asymmetric-key* [name]
        |  +--rw name                                    string
        |  +--rw algorithm
        |  |       ct:key-algorithm-ref
        |  +--rw public-key                              binary
        |  +--rw private-key                             union
        |  +--rw certificates
        |  |  +--rw certificate* [name]
        |  |     +--rw name                      string
        |  |     +--rw cert
        |  |     |       ct:end-entity-cert-cms
        |  |     +---n certificate-expiration
        |  |        +-- expiration-date?   yang:date-and-time
        |  +---x generate-certificate-signing-request
        |     +---w input
        |     |  +---w subject       binary
        |     |  +---w attributes?   binary
        |     +--ro output
        |        +--ro certificate-signing-request    binary
        +---x generate-asymmetric-key
           +---w input
              +---w name         string
              +---w algorithm    ct:key-algorithm-ref

  grouping end-entity-cert-grouping
    +-- cert                      ct:end-entity-cert-cms
    +---n certificate-expiration
       +-- expiration-date?   yang:date-and-time
  grouping local-or-keystore-end-entity-certificate-grouping
    +-- (local-or-keystore)
       +--:(local)
       |  +-- algorithm                ct:key-algorithm-ref
       |  +-- public-key               binary
       |  +-- private-key              union
```

```
          |  +-- cert                      ct:end-entity-cert-cms
          |  +---n certificate-expiration
          |     +-- expiration-date?   yang:date-and-time
          +--:(keystore) {keystore-implemented}?
             +-- reference
                     ks:asymmetric-key-certificate-ref
     grouping local-or-keystore-asymmetric-key-with-certs-grouping
       +-- (local-or-keystore)
          +--:(local)
          |  +-- algorithm
          |  |       ct:key-algorithm-ref
          |  +-- public-key                              binary
          |  +-- private-key                             union
          |  +-- certificates
          |  |  +-- certificate* [name]
          |  |     +-- name?                     string
          |  |     +-- cert                      ct:end-entity-cert-cms
          |  |     +---n certificate-expiration
          |  |        +-- expiration-date?   yang:date-and-time
          |  +---x generate-certificate-signing-request
          |     +---w input
          |     |  +---w subject       binary
          |     |  +---w attributes?   binary
          |     +--ro output
          |        +--ro certificate-signing-request    binary
          +--:(keystore) {keystore-implemented}?
             +-- reference
                     ks:asymmetric-key-ref
     grouping trust-anchor-cert-grouping
       +-- cert    ct:trust-anchor-cert-cms
     grouping asymmetric-key-pair-grouping
       +-- algorithm      ct:key-algorithm-ref
       +-- public-key     binary
       +-- private-key    union
     grouping public-key-grouping
       +-- algorithm     ct:key-algorithm-ref
       +-- public-key    binary
     grouping asymmetric-key-pair-with-certs-grouping
       +-- algorithm                              ct:key-algorithm-ref
       +-- public-key                             binary
       +-- private-key                            union
       +-- certificates
       |  +-- certificate* [name]
       |     +-- name?                     string
       |     +-- cert                      ct:end-entity-cert-cms
       |     +---n certificate-expiration
       |        +-- expiration-date?   yang:date-and-time
       +---x generate-certificate-signing-request
```

```
          +---w input
          |  +---w subject       binary
          |  +---w attributes?   binary
          +--ro output
             +--ro certificate-signing-request    binary
     grouping local-or-keystore-asymmetric-key-grouping
       +-- (local-or-keystore)
          +--:(local)
          |  +-- algorithm      ct:key-algorithm-ref
          |  +-- public-key     binary
          |  +-- private-key    union
          +--:(keystore) {keystore-implemented}?
             +-- reference      ks:asymmetric-key-ref
```

3.2. Example Usage

The following example illustrates what a fully configured keystore might look like in <operational>, as described by Section 5.3 in [RFC8342]. This datastore view illustrates data set by the manufacturing process alongside conventional configuration. This keystore instance has three keys, two having one associated certificate and one having two associated certificates.

```
<keystore xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore"
          xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin"
          xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">
  <asymmetric-keys>

    <asymmetric-key or:origin="or:intended">
      <name>ex-rsa-key</name>
      <algorithm>ct:rsa1024</algorithm>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <certificates>
        <certificate>
          <name>ex-rsa-cert</name>
          <cert>base64encodedvalue==</cert>
        </certificate>
      </certificates>
    </asymmetric-key>

    <asymmetric-key or:origin="or:intended">
      <name>tls-ec-key</name>
      <algorithm>ct:secp256r1</algorithm>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <certificates>
        <certificate>
```

```
            <name>tls-ec-cert</name>
            <cert>base64encodedvalue==</cert>
          </certificate>
        </certificates>
      </asymmetric-key>

      <asymmetric-key or:origin="or:system">
        <name>tpm-protected-key</name>
        <algorithm>ct:rsa2048</algorithm>
        <private-key>hardware-protected</private-key>
        <public-key>base64encodedvalue==</public-key>
        <certificates>
          <certificate>
            <name>builtin-idevid-cert</name>
            <cert>base64encodedvalue==</cert>
          </certificate>
          <certificate or:origin="or:intended">
            <name>my-ldevid-cert</name>
            <cert>base64encodedvalue==</cert>
          </certificate>
        </certificates>
      </asymmetric-key>

    </asymmetric-keys>
  </keystore>
```

The following example illustrates the "generate-private-key" 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">
      <asymmetric-keys>
        <generate-asymmetric-key>
          <name>ex-key-sect571r1</name>
          <algorithm
           xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">
              ct:secp521r1
          </algorithm>
        </generate-asymmetric-key>
      </asymmetric-keys>
    </keystore>
  </action>
</rpc>

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

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">
      <asymmetric-keys>
        <asymmetric-key>
          <name>ex-key-sect571r1</name>
          <generate-certificate-signing-request>
            <subject>base64encodedvalue==</subject>
            <attributes>base64encodedvalue==</attributes>
          </generate-certificate-signing-request>
        </asymmetric-key>
      </asymmetric-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">
     base64encodedvalue==
   </certificate-signing-request>
</rpc-reply>
```

The following example illustrates the "certificate-expiration" notification in use with the NETCONF protocol.

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2018-05-25T00:01:00Z</eventTime>
  <keystore xmlns="urn:ietf:params:xml:ns:yang:ietf-keystore">
    <asymmetric-keys>
      <asymmetric-key>
        <name>tpm-protected-key</name>
        <certificates>
          <certificate>
            <name>my-ldevid-cert</name>
            <certificate-expiration>
              <expiration-date>
                2018-08-05T14:18:53-05:00
              </expiration-date>
            </certificate-expiration>
          </certificate>
        </certificates>
      </asymmetric-key>
    </asymmetric-keys>
  </keystore>
</notification>
```

The following example module has been constructed to illustrate the "local-or-keystore-asymmetric-key-grouping" grouping defined in the "ietf-keystore" module.

```
module ex-keystore-usage {
  yang-version 1.1;

  namespace "http://example.com/ns/example-keystore-usage";
  prefix "eku";

  import ietf-keystore {
    prefix ks;
    reference
      "RFC VVVV: YANG Data Model for a 'Keystore' Mechanism";
  }

  organization
   "Example Corporation";

  contact
   "Author: YANG Designer <mailto:yang.designer@example.com>";

  description
   "This module illustrates the grouping defined in the keystore
    draft called 'local-or-keystore-asymmetric-key-grouping'.";

  revision "YYYY-MM-DD" {
    description
     "Initial version";
    reference
     "RFC XXXX: YANG Data Model for a 'Keystore' Mechanism";
  }

  container keys {
    description
      "A container of keys.";
    list key {
      key name;
      leaf name {
        type string;
        description
          "An arbitrary name for this key.";
      }
      uses ks:local-or-keystore-asymmetric-key-grouping;
      description
        "A key which may be configured locally or be a reference to
         a key in the keystore.";
    }
  }
}
```

The following example illustrates what two configured keys, one local and the other remote, might look like. This example consistent with other examples above (i.e., the referenced key is in an example above).

```
<keys xmlns="http://example.com/ns/example-keystore-usage">
  <key>
    <name>locally-defined key</name>
    <algorithm
      xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-types">
      ct:secp521r1
    </algorithm>
    <private-key>base64encodedvalue==</private-key>
    <public-key>base64encodedvalue==</public-key>
  </key>
  <key>
    <name>keystore-defined key</name>
    <reference>ex-rsa-key</reference>
  </key>
</keys>
```

3.3. YANG Module

This YANG module imports modules defined in [RFC6536], [RFC6991], and [I-D.ietf-netconf-crypto-types]. This module uses data types defined in [RFC2986], [RFC3447], [RFC5652], [RFC5915], [RFC6125], and [ITU.X690.2015].

```
<CODE BEGINS> file "ietf-keystore@2018-06-04.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-crypto-types {
    prefix ct;
    reference
      "RFC CCCC: Common YANG Data Types for Cryptography";
  }

  organization
```

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

     contact
      "WG Web:   <http://datatracker.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) 2018 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 "2018-06-04" {
       description
        "Initial version";
       reference
        "RFC VVVV: YANG Data Model for a 'Keystore' Mechanism";
     }

     // Features

     feature keystore-implemented {
       description
        "The 'keystore-implemented' feature indicates that the server
         implements the keystore, and therefore groupings defined in
         this module that reference the keystore are usable.";
     }


     // Typedefs

     typedef asymmetric-key-ref {
       type leafref {
```

```
        path "/ks:keystore/ks:asymmetric-keys/ks:asymmetric-key"
             + "/ks:name";
        require-instance false;
      }
      description
        "This typedef enables modules to easily define a reference
         to an asymmetric key stored in the keystore. The require
         instance attribute is false to enable the referencing of
         asymmetric keys that exist only in <operational>.";
      reference
        "RFC 8342: Network Management Datastore Architecture (NMDA)";
    }

    typedef asymmetric-key-certificate-ref {
      type leafref {
        path "/ks:keystore/ks:asymmetric-keys/ks:asymmetric-key"
             + "/ks:certificates/ks:certificate/ks:name";
        require-instance false;
      }
      description
        "This typedef enables modules to easily define a reference
         to a specific certificate associated with an asymmetric key
         stored in the keystore.  The require instance attribute is
         false to enable the referencing of certificates that exist
         only in <operational>.";
      reference
        "RFC 8342: Network Management Datastore Architecture (NMDA)";
    }


    // Groupings
    //
    // These groupings are factored out more than needed for
    // reusability purposes.

    grouping public-key-grouping {
      description
        "A public key.";
      leaf algorithm {
        type ct:key-algorithm-ref;
        mandatory true;
        description
          "Identifies the key's algorithm.  More specifically,
           this leaf specifies how the 'public-key' binary leaf
           is encoded.";
        reference
          "RFC CCCC: Common YANG Data Types for Cryptography";
      }
```

```
        leaf public-key {
          type binary;
          mandatory true;
          description
            "A binary that contains the value of the public key.  The
             interpretation of the content is defined by the key
             algorithm.  For example, a DSA key is an integer, an RSA
             key is represented as RSAPublicKey as defined in
             RFC 3447, and an Elliptic Curve Cryptography (ECC) key
             is represented using the 'publicKey' described in
             RFC 5915.";
          reference
            "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
                       RSA Cryptography Specifications Version 2.1.
             RFC 5915: Elliptic Curve Private Key Structure.";
        }
      }

      grouping asymmetric-key-pair-grouping {
        description
          "A private/public key pair.";
        uses public-key-grouping;
        leaf private-key {
          type union {
            type binary;
            type enumeration {
              enum "hardware-protected" {
                description
                 "The private key is inaccessible due to being
                  protected by a cryptographic hardware module
                  (e.g., a TPM).";
              }
            }
          }
          mandatory true;
          description
            "A binary that contains the value of the private key.  The
             interpretation of the content is defined by the key
             algorithm.  For example, a DSA key is an integer, an RSA
             key is represented as RSAPrivateKey as defined in
             RFC 3447, and an Elliptic Curve Cryptography (ECC) key
             is represented as ECPrivateKey as defined in RFC 5915.";
          reference
            "RFC 3447: Public-Key Cryptography Standards (PKCS) #1:
                       RSA Cryptography Specifications Version 2.1.
             RFC 5915: Elliptic Curve Private Key Structure.";
        }
      }
```

```
     grouping trust-anchor-cert-grouping {
       description
        "A certificate, and a notification for when it might expire.";
       leaf cert {
         type ct:trust-anchor-cert-cms;
         mandatory true;
         description
          "The binary certificate data for this certificate.";
         reference
          "RFC YYYY: Common YANG Data Types for Cryptography";
       }
     }

     grouping end-entity-cert-grouping {
       description
        "A certificate, and a notification for when it might expire.";
       leaf cert {
         type ct:end-entity-cert-cms;
         mandatory true;
         description
          "The binary certificate data for this certificate.";
         reference
           "RFC YYYY: Common YANG Data Types for Cryptography";
       }
       notification certificate-expiration {
         description
           "A notification indicating that the 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 until the issue is resolved.";
         leaf expiration-date {
           type yang:date-and-time;
           //mandatory true;
           description
             "Identifies the expiration date on the certificate.";
         }
       }
     }

     grouping asymmetric-key-pair-with-certs-grouping {
       description
         "A private/public key pair and associated certificates.";
       uses asymmetric-key-pair-grouping;
       container certificates {
         description
           "Certificates associated with this asymmetric key.
```

```
            More than one certificate supports, for instance,
            a TPM-protected asymmetric key that has both IDevID
            and LDevID certificates associated.";
          list certificate {
            key name;
            description
              "A certificate for this asymmetric key.";
            leaf name {
              type string;
              description
                "An arbitrary name for the certificate.";
            }
            uses end-entity-cert-grouping;
          } // end certifcate
        } // end certificates
        action generate-certificate-signing-request {
          description
            "Generates a certificate signing request structure for
             the associated asymmetric 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 per 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 Syntaxi
                             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 structure
                 CertificationRequestInfo 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 4.2 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 output
      } // end generate-certificate-signing-request
    }

    grouping local-or-keystore-asymmetric-key-grouping {
      description
        "A grouping that expands to allow the key to be either stored
         locally within the using data model, or be a reference to an
         asymmetric key stored in the keystore.";
      choice local-or-keystore {
        mandatory true;
        case local {
          uses asymmetric-key-pair-grouping;
        }
        case keystore {
```

```
          if-feature "keystore-implemented";
          leaf reference {
            type ks:asymmetric-key-ref;
            mandatory true;
            description
              "A reference to a value that exists in the keystore.";
          }
        }
        description
          "A choice between an inlined definition and a definition
           that exists in the keystore.";
      }
    }

    grouping local-or-keystore-asymmetric-key-with-certs-grouping {
      description
        "A grouping that expands to allow the key to be either stored
         locally within the using data model, or be a reference to an
         asymmetric key stored in the keystore.";
      choice local-or-keystore {
        mandatory true;
        case local {
          uses asymmetric-key-pair-with-certs-grouping;
        }
        case keystore {
          if-feature "keystore-implemented";
          leaf reference {
            type ks:asymmetric-key-ref;
            mandatory true;
            description
              "A reference to a value that exists in the keystore.";
          }
        }
        description
          "A choice between an inlined definition and a definition
           that exists in the keystore.";
      }
    }

    grouping local-or-keystore-end-entity-certificate-grouping {
      description
        "A grouping that expands to allow the end-entity certificate
         (and the associated private key) to be either stored locally
         within the using data model, or be a reference to a specific
         certificate in the keystore.";
      choice local-or-keystore {
        mandatory true;
        case local {
```

```
          uses ks:asymmetric-key-pair-grouping;
          uses ks:end-entity-cert-grouping;
        }
        case keystore {
          if-feature "keystore-implemented";
          leaf reference {
            type ks:asymmetric-key-certificate-ref;
            mandatory true;
            description
              "A reference to a value that exists in the keystore.";
          }
        }
        description
          "A choice between an inlined definition and a definition
           that exists in the keystore.";
      }
    }


    // protocol accessible nodes

    container keystore {
      description
        "The keystore contains a list of keys.";

      container asymmetric-keys {
        description
          "A list of asymmetric keys.";
        list asymmetric-key {
          key name;
          description
            "An asymmetric key.";
          leaf name {
            type string;
            description
              "An arbitrary name for the asymmetric key.";
          }
          uses asymmetric-key-pair-with-certs-grouping;
        } // end asymmetric-key

        action generate-asymmetric-key {
          description
            "Requests the device to generate an asymmetric key using
             the specified asymmetric key algorithm.  This action is
             primarily to support cryptographic processors that must
             generate the asymmetric key themselves.  The resulting
             asymmetric key is considered operational state and hence
             present only in <operational>.";
```

```
          input {
            leaf name {
              type string;
              mandatory true;
              description
                "The name the asymmetric key should have when listed
                 in /keystore/asymmetric-keys/asymmetric-key, in
                 <operational>.";
            }
            leaf algorithm {
              type ct:key-algorithm-ref;
              mandatory true;
              description
                "The algorithm to be used when generating the
                 asymmetric key.";
              reference
                "RFC CCCC: Common YANG Data Types for Cryptography";
            }
          }
        } // end generate-asymmetric-key
      } // end asymmetric-keys
    } // end keystore


  }
  <CODE ENDS>
```

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. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated 'recovery session'.

/keystore/asymmetric-keys/asymmetric-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 then 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/asymmetric-keys/asymmetric-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. However, no NACM annotations are applied as the data SHOULD be editable by users other than a designated 'recovery session'.

Some of the 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 action, 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.

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 is was unclear if there was market demand for it. 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.

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:       ks
reference:    RFC VVVV
```

6. References

6.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K., "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-00 (work in progress), June 2018.

[ITU.X690.2015]
International Telecommunication Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1, August 2015, <https://www.itu.int/rec/T-REC-X.690/>.

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

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, DOI 10.17487/RFC3447, February 2003, <https://www.rfc-editor.org/info/rfc3447>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5915] Turner, S. and D. Brown, "Elliptic Curve Private Key Structure", RFC 5915, DOI 10.17487/RFC5915, June 2010, <https://www.rfc-editor.org/info/rfc5915>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://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, <https://www.rfc-editor.org/info/rfc6536>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://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, <https://www.rfc-editor.org/info/rfc7950>.

6.2. Informative References

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc5056>.

[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, <https://www.rfc-editor.org/info/rfc6125>.

[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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://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, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[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. 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.2. 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.

A.3. 02 to 03

- o Changed draft's title.
- o Added missing references.
- o Collapsed sections and levels.
- o Added RFC 8174 to Requirements Language Section.
- o Renamed 'trusted-certificates' to 'pinned-certificates'.
- o Changed 'public-key' from config false to config true.
- o Switched 'host-key' from OneAsymmetricKey to definition from RFC 4253.

A.4. 03 to 04

- o Added typedefs around leafrefs to common keystore paths
- o Now tree diagrams reference ietf-netmod-yang-tree-diagrams
- o Removed Design Considerations section
- o Moved key and certificate definitions from data tree to groupings

A.5. 04 to 05

- o FIXME
- o FIXME
- o FIXME

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, Mahesh Jethanandani, Radek Krejci, Reshad Rahman, Tom Petch, Juergen Schoenwaelder, Phil Shafer, Sean Turner, Eric Voit, Bert Wijnen, and Liang Xia.

Author’s Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 6, 2018

K. Watsen
Juniper Networks
G. Wu
Cisco Networks

June 4, 2018

NETCONF Client and Server Models
draft-ietf-netconf-netconf-client-server-06

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:

- I-D.ietf-netconf-keystore

- I-D.ietf-netconf-ssh-client-server

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

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

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

- "YYYY" --> the assigned RFC value for I-D.ietf-netconf-ssh-client-server

- "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 "2018-06-04" --> 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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. The NETCONF Client Model 4
  3.1. Tree Diagram 4
  3.2. Example Usage 9
  3.3. YANG Module 11
4. The NETCONF Server Model 22
  4.1. Tree Diagram 22

4.2. Example Usage 28
4.3. YANG Module 33
5. Design Considerations 44
5.1. Support all NETCONF transports 45
5.2. Enable each transport to select which keys to use 45
5.3. Support authenticating NETCONF clients certificates . . . 45
5.4. Support mapping authenticated NETCONF client certificates to usernames 45
5.5. Support both listening for connections and call home . . 45
5.6. For Call Home connections 46
5.6.1. Support more than one NETCONF client 46
5.6.2. Support NETCONF clients having more than one endpoint 46
5.6.3. Support a reconnection strategy 46
5.6.4. Support both persistent and periodic connections . . 46
5.6.5. Reconnection strategy for periodic connections . . . 46
5.6.6. Keep-alives for persistent connections 47
5.6.7. Customizations for periodic connections 47
6. Security Considerations 47
7. IANA Considerations 48
7.1. The IETF XML Registry 48
7.2. The YANG Module Names Registry 48
8. References 49
8.1. Normative References 49
8.2. Informative References 50
Appendix A. Change Log 51
A.1. 00 to 01 51
A.2. 01 to 02 51
A.3. 02 to 03 51
A.4. 03 to 04 51
A.5. 04 to 05 51
A.6. 05 to 06 52
Acknowledgements 52
Authors' Addresses 52

1. Introduction

This document defines two YANG [RFC7950] modules, one module to configure a NETCONF [RFC6241] client and the other module to configure a NETCONF server. Both modules support both NETCONF over SSH [RFC6242] and NETCONF over TLS [RFC7589] and NETCONF Call Home connections [RFC8071].

2. 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.

3. 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.

3.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-netconf-client" module. Just the container is displayed below, but there is also a reuable grouping by the same name that the container is using.

[Note: '\' line wrapping for formatting only]

```
module: ietf-netconf-client
  +--rw netconf-client
     +--rw initiate! {initiate}?
     |  +--rw netconf-server* [name]
     |     +--rw name                 string
     |     +--rw endpoints
     |     |  +--rw endpoint* [name]
     |     |     +--rw name         string
     |     |     +--rw (transport)
     |     |        +--:(ssh) {ssh-initiate}?
     |     |        |  +--rw ssh
     |     |        |     +--rw address?            inet:host
     |     |        |     +--rw port?               inet:port-number\

     |     |        |     +--rw client-identity
     |     |        |     |  +--rw username?            string
     |     |        |     |  +--rw (auth-type)
     |     |        |     |     +--:(password)
```

```
        |     |        |     |     |  +--rw password?      string
        |     |        |     |     +--:(public-key)
        |     |        |     |     |  +--rw public-key
        |     |        |     |     |     +--rw (local-or-keystore)
        |     |        |     |     |        +--:(local)
        |     |        |     |     |        |  +--rw algorithm
        |     |        |     |     |        |  |       ct:key-algorithm\
   -ref
        |     |        |     |     |        |  +--rw public-key     bin\
   ary
        |     |        |     |     |        |  +--rw private-key    uni\
   on
        |     |        |     |     |        +--:(keystore)
        |     |        |     |     |                 {keystore-implemen\
   ted}?
        |     |        |     |     |           +--rw reference
        |     |        |     |     |                   ks:asymmetric-ke\
   y-ref
        |     |        |     |     +--:(certificate)
        |     |        |     |        +--rw certificate
        |     |        |     |                {sshcmn:ssh-x509-certs}?
        |     |        |     |           +--rw (local-or-keystore)
        |     |        |     |              +--:(local)
        |     |        |     |              |  +--rw algorithm
        |     |        |     |              |  |       ct:key-algorithm\
   -ref
        |     |        |     |              |  +--rw public-key
        |     |        |     |              |  |       binary
        |     |        |     |              |  +--rw private-key
        |     |        |     |              |  |       union
        |     |        |     |              |  +--rw cert
        |     |        |     |              |  |       ct:end-entity-ce\
   rt-cms
        |     |        |     |              |  +---n certificate-expira\
   tion
        |     |        |     |              |     +-- expiration-date?
        |     |        |     |              |             yang:date-and\
   -time
        |     |        |     |              +--:(keystore)
        |     |        |     |                       {keystore-implemen\
   ted}?
        |     |        |     |                 +--rw reference
        |     |        |     |                         ks:asymmetric-ke\
   y-certificate-ref
        |     |        |     +--rw server-auth
        |     |        |     |  +--rw pinned-ssh-host-keys?
        |     |        |     |  |       ta:pinned-host-keys-ref
        |     |        |     |  +--rw pinned-ca-certs?
```

```
        |     |        |     |  |       ta:pinned-certificates-ref
        |     |        |     |  |       {sshcmn:ssh-x509-certs}?
        |     |        |     |  +--rw pinned-server-certs?
        |     |        |     |          ta:pinned-certificates-ref
        |     |        |     |          {sshcmn:ssh-x509-certs}?
        |     |        |     +--rw transport-params
        |     |        |             {ssh-client-transport-params-confi\
   g}?
        |     |        |        +--rw host-key
        |     |        |        |  +--rw host-key-alg*   identityref
        |     |        |        +--rw key-exchange
        |     |        |        |  +--rw key-exchange-alg*   identityre\
   f
        |     |        |        +--rw encryption
        |     |        |        |  +--rw encryption-alg*   identityref
        |     |        |        +--rw mac
        |     |        |           +--rw mac-alg*   identityref
        |     |        +--:(tls) {tls-initiate}?
        |     |           +--rw tls
        |     |              +--rw address?           inet:host
        |     |              +--rw port?              inet:port-number
        |     |              +--rw client-identity
        |     |              |  +--rw (auth-type)
        |     |              |     +--:(certificate)
        |     |              |        +--rw certificate
        |     |              |           +--rw (local-or-keystore)
        |     |              |              +--:(local)
        |     |              |              |  +--rw algorithm
        |     |              |              |  |       ct:key-algorithm\
   -ref
        |     |              |              |  +--rw public-key
        |     |              |              |  |       binary
        |     |              |              |  +--rw private-key
        |     |              |              |  |       union
        |     |              |              |  +--rw cert
        |     |              |              |  |       ct:end-entity-ce\
   rt-cms
        |     |              |              |  +---n certificate-expira\
   tion
        |     |              |              |     +-- expiration-date?
        |     |              |              |             yang:date-and\
   -time
        |     |              |              +--:(keystore)
        |     |              |                       {keystore-implemen\
   ted}?
        |     |              |                 +--rw reference
        |     |              |                         ks:asymmetric-ke\
   y-certificate-ref
```

```
        |     |             +--rw server-auth
        |     |             |  +--rw pinned-ca-certs?
        |     |             |  |       ta:pinned-certificates-ref
        |     |             |  +--rw pinned-server-certs?
        |     |             |          ta:pinned-certificates-ref
        |     |             +--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 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 client-identity
                |     |  +--rw username?            string
                |     |  +--rw (auth-type)
                |     |     +--:(password)
                |     |     |  +--rw password?      string
                |     |     +--:(public-key)
                |     |     |  +--rw public-key
                |     |     |     +--rw (local-or-keystore)
                |     |     |        +--:(local)
                |     |     |        |  +--rw algorithm
                |     |     |        |  |       ct:key-algorithm-ref
                |     |     |        |  +--rw public-key     binary
                |     |     |        |  +--rw private-key    union
```

```
                  |     |     |        +--:(keystore)
                  |     |     |                 {keystore-implemented}?
                  |     |     |           +--rw reference
                  |     |     |                   ks:asymmetric-key-ref
                  |     |     +--:(certificate)
                  |     |        +--rw certificate {sshcmn:ssh-x509-cert\
   s}?
                  |     |           +--rw (local-or-keystore)
                  |     |              +--:(local)
                  |     |              |  +--rw algorithm
                  |     |              |  |       ct:key-algorithm-ref
                  |     |              |  +--rw public-key
                  |     |              |  |       binary
                  |     |              |  +--rw private-key
                  |     |              |  |       union
                  |     |              |  +--rw cert
                  |     |              |  |       ct:end-entity-cert-cms\

                  |     |              |  +---n certificate-expiration
                  |     |              |     +-- expiration-date?
                  |     |              |             yang:date-and-time
                  |     |              +--:(keystore)
                  |     |                       {keystore-implemented}?
                  |     |                 +--rw reference
                  |     |                         ks:asymmetric-key-cert\
   ificate-ref
                  |     +--rw server-auth
                  |     |  +--rw pinned-ssh-host-keys?
                  |     |  |       ta:pinned-host-keys-ref
                  |     |  +--rw pinned-ca-certs?
                  |     |  |       ta:pinned-certificates-ref
                  |     |  |       {sshcmn:ssh-x509-certs}?
                  |     |  +--rw pinned-server-certs?
                  |     |          ta:pinned-certificates-ref
                  |     |          {sshcmn:ssh-x509-certs}?
                  |     +--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
                  +--:(tls) {tls-listen}?
                     +--rw tls
                        +--rw address?           inet:ip-address
```

```
                        +--rw port?              inet:port-number
                        +--rw client-identity
                        |  +--rw (auth-type)
                        |     +--:(certificate)
                        |        +--rw certificate
                        |           +--rw (local-or-keystore)
                        |              +--:(local)
                        |              |  +--rw algorithm
                        |              |  |       ct:key-algorithm-ref
                        |              |  +--rw public-key
                        |              |  |       binary
                        |              |  +--rw private-key
                        |              |  |       union
                        |              |  +--rw cert
                        |              |  |       ct:end-entity-cert-cms\

                        |              |  +---n certificate-expiration
                        |              |     +-- expiration-date?
                        |              |             yang:date-and-time
                        |              +--:(keystore)
                        |                       {keystore-implemented}?
                        |                 +--rw reference
                        |                         ks:asymmetric-key-cert\
   ificate-ref
                        +--rw server-auth
                        |  +--rw pinned-ca-certs?
                        |  |       ta:pinned-certificates-ref
                        |  +--rw pinned-server-certs?
                        |          ta:pinned-certificates-ref
                        +--rw hello-params
                                {tls-client-hello-params-config}?
                           +--rw tls-versions
                           |  +--rw tls-version*   identityref
                           +--rw cipher-suites
                              +--rw cipher-suite*   identityref
```

3.2. 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].

[Note: '\' line wrapping for formatting only]

```
<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>
      <endpoints>
        <endpoint>
          <name>corp-fw1.example.com</name>
          <ssh>
            <address>corp-fw1.example.com</address>
            <client-identity>
              <username>foobar</username>
              <public-key>
                <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:iet\
f-crypto-types">ct:secp521r1</algorithm>
                <private-key>base64encodedvalue==</private-key>
                <public-key>base64encodedvalue==</public-key>
              </public-key>
            </client-identity>
            <server-auth>
              <pinned-ca-certs>explicitly-trusted-server-ca-certs</p\
inned-ca-certs>
              <pinned-server-certs>explicitly-trusted-server-certs</\
pinned-server-certs>
            </server-auth>
          </ssh>
        </endpoint>
        <endpoint>
          <name>corp-fw2.example.com</name>
          <ssh>
            <address>corp-fw2.example.com</address>
            <client-identity>
              <username>foobar</username>
              <public-key>
                <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:iet\
f-crypto-types">ct:secp521r1</algorithm>
                <private-key>base64encodedvalue==</private-key>
                <public-key>base64encodedvalue==</public-key>
              </public-key>
            </client-identity>
            <server-auth>
              <pinned-ca-certs>explicitly-trusted-server-ca-certs</p\
inned-ca-certs>
              <pinned-server-certs>explicitly-trusted-server-certs</\
pinned-server-certs>
            </server-auth>
```

```
            </ssh>
          </endpoint>
        </endpoints>
        <connection-type>
          <persistent/>
        </connection-type>
        <reconnect-strategy>
          <start-with>last-connected</start-with>
        </reconnect-strategy>
      </netconf-server>
    </initiate>

    <!-- endpoints to listen for NETCONF Call Home connections on -->
    <listen>
      <endpoint>
        <name>Intranet-facing listener</name>
        <ssh>
          <address>192.0.2.7</address>
          <client-identity>
            <username>foobar</username>
            <public-key>
              <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-cr\
ypto-types">ct:secp521r1</algorithm>
              <private-key>base64encodedvalue==</private-key>
              <public-key>base64encodedvalue==</public-key>
            </public-key>
          </client-identity>
          <server-auth>
            <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinne\
d-ca-certs>
            <pinned-server-certs>explicitly-trusted-server-certs</pinn\
ed-server-certs>
            <pinned-ssh-host-keys>explicitly-trusted-ssh-host-keys</pi\
nned-ssh-host-keys>
          </server-auth>
        </ssh>
      </endpoint>
    </listen>
  </netconf-client>
```

3.3. YANG Module

This YANG module has normative references to [RFC6242], [RFC6991], [RFC7589], [RFC8071], [I-D.ietf-netconf-ssh-client-server], and [I-D.ietf-netconf-tls-client-server].

```
<CODE BEGINS> file "ietf-netconf-client@2018-06-04.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 2018-06-04; // stable grouping definitions
  reference
    "RFC YYYY: YANG Groupings for SSH Clients and SSH Servers";
}

import ietf-tls-client {
  prefix ts;
  revision-date 2018-06-04; // stable grouping definitions
  reference
    "RFC ZZZZ: YANG Groupings for TLS Clients and TLS Servers";
}

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

contact
 "WG Web:   <http://datatracker.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) 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 "2018-06-04" {
       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;
         presence "Enables client to initiate TCP connections";
         description
           "Configures client initiating underlying TCP connections.";
         list netconf-server {
           key name;
           min-elements 1;
           description
             "List of NETCONF servers the NETCONF client is to
              initiate connections to in parallel.";
           leaf name {
             type string;
             description
               "An arbitrary name for the NETCONF server.";
           }
           container endpoints {
```

```
              description
                "Container for the list of endpoints.";
              list endpoint {
                key name;
                min-elements 1;
                ordered-by user;
                description
                  "A user-ordered list of endpoints that the NETCONF
                   client will attempt to connect to in the specified
                   sequence.  Defining more than one enables
                   high-availability.";
                leaf name {
                  type string;
                  description
                    "An arbitrary name for the endpoint.";
                }
                choice transport {
                  mandatory true;
                  description
                    "Selects between available transports.";
                  case ssh {
                    if-feature ssh-initiate;
                    container ssh {
                      description
                        "Specifies IP and SSH specific configuration
                         for the connection.";
                      leaf address {
                        type inet:host;
                        description
                         "The IP address or hostname of the endpoint.
                          If a domain name is configured, then the
                          DNS resolution should happen on each usage
                          attempt.  If the DNS resolution results in
                          multiple IP addresses, the IP addresses will
                          be tried according to local preference order
                          until a connection has been established or
                          until all IP addresses have failed.";
                      }
                      leaf port {
                        type inet:port-number;
                        default 830;
                        description
                          "The IP port for this endpoint.  The NETCONF
                           client will use the IANA-assigned well-known
                           port for 'netconf-ssh' (830) if no value is
                           specified.";
                      }
                      uses ss:ssh-client-grouping;
```

```
                  }
                } // end ssh
                case tls {
                  if-feature tls-initiate;
                  container tls {
                    description
                      "Specifies IP and TLS specific configuration
                       for the connection.";
                    leaf address {
                      type inet:host;
                      description
                        "The IP address or hostname of the endpoint.
                         If a domain name is configured, then the
                         DNS resolution should happen on each usage
                         attempt.  If the DNS resolution results in
                         multiple IP addresses, the IP addresses will
                         be tried according to local preference order
                         until a connection has been established or
                         until all IP addresses have failed.";
                      }
                      leaf port {
                        type inet:port-number;
                        default 6513;
                        description
                          "The IP port for this endpoint. The NETCONF
                           client will use the IANA-assigned well-
                           known port for 'netconf-tls' (6513) if no
                           value is specified.";
                      }
                      uses ts:tls-client-grouping {
                        refine "client-identity/auth-type" {
                        mandatory true;
                         description
                           "NETCONF/TLS clients MUST pass some
                            authentication credentials.";
                       }
                     }
                   }
                 } // end tls
               }
             }
           }

           container connection-type {
             description
              "Indicates the kind of connection to use.";
             choice connection-type {
               default persistent-connection;
```

```
                description
                  "Selects between available connection types.";
                case persistent-connection {
                  container persistent {
                    presence
                     "Indicates that a persistent connection is to be
                      maintained.";
                    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 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
                   "Indicates that a periodic connection is to be
                    maintained.";
                  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 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 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.   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;
         presence "Enables client to accept call-home connections";
         description
           "Configures client accepting call-home TCP connections.";

         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;
           min-elements 1;
           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 on for incoming call-
                      home connections.  The NETCONF client will listen
                      on all configured interfaces if no value is
                      specified.  INADDR_ANY (0.0.0.0) or INADDR6_ANY
                      (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
                      the server is to listen on all IPv4 or IPv6
                      addresses, respectively.";
                  }
                  leaf port {
                    type inet:port-number;
                    default 4334;
                    description
                     "The port number to listen on for call-home
                      connections.  The NETCONF client will listen
                      on the IANA-assigned well-known port for
                      'netconf-ch-ssh' (4334) if no value is
                      specified.";
                  }
                  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 on for incoming call-
                      home connections.  The NETCONF client will listen
                      on all configured interfaces if no value is
                      specified.  INADDR_ANY (0.0.0.0) or INADDR6_ANY
                      (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
                      the server is to listen on all IPv4 or IPv6
                      addresses, respectively.";
                  }
                  leaf port {
                    type inet:port-number;
```

```
                    default 4335;
                    description
                     "The port number to listen on for call-home
                      connections.  The NETCONF client will listen
                      on the IANA-assigned well-known port for
                      'netconf-ch-tls' (4335) if no value is
                      specified.";
                  }
                  uses ts:tls-client-grouping {
                    refine "client-identity/auth-type" {
                      mandatory true;
                      description
                        "NETCONF/TLS clients MUST pass some
                         authentication credentials.";
                    }
                  }
                }
              }
            } // end transport
          } // end endpoint
        } // end listen

      } // end netconf-client
    }
    <CODE ENDS>
```

4. 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.

4.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-netconf-server" module. Just the container is

displayed below, but there is also a reuable grouping by the same name that the container is using.

[Note: '\' line wrapping for formatting only]

```
module: ietf-netconf-server
  +--rw netconf-server
     +--rw listen! {listen}?
     |  +--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-identity
     |        |     |  +--rw host-key* [name]
     |        |     |     +--rw name                 string
     |        |     |     +--rw (host-key-type)
     |        |     |        +--:(public-key)
     |        |     |        |  +--rw public-key
     |        |     |        |     +--rw (local-or-keystore)
     |        |     |        |        +--:(local)
     |        |     |        |        |  +--rw algorithm
     |        |     |        |        |  |       ct:key-algorithm-re\
f
     |        |     |        |        |  +--rw public-key     binary\

     |        |     |        |        |  +--rw private-key    union
     |        |     |        |        +--:(keystore)
     |        |     |        |                 {keystore-implemented\
}?
     |        |     |        |           +--rw reference
     |        |     |        |                   ks:asymmetric-key-r\
ef
     |        |     |        +--:(certificate)
     |        |     |           +--rw certificate
     |        |     |                   {sshcmn:ssh-x509-certs}?
     |        |     |              +--rw (local-or-keystore)
     |        |     |                 +--:(local)
     |        |     |                 |  +--rw algorithm
     |        |     |                 |  |       ct:key-algorithm-re\
f
     |        |     |                 |  +--rw public-key
     |        |     |                 |  |       binary
     |        |     |                 |  +--rw private-key
```

```
        |        |     |                 |  |       union
        |        |     |                 |  +--rw cert
        |        |     |                 |  |       ct:end-entity-cert-\
   cms
        |        |     |                 |  +---n certificate-expiratio\
   n
        |        |     |                 |     +-- expiration-date?
        |        |     |                 |             yang:date-and-ti\
   me
        |        |     |                 +--:(keystore)
        |        |     |                          {keystore-implemented\
   }?
        |        |     |                    +--rw reference
        |        |     |                            ks:asymmetric-key-c\
   ertificate-ref
        |        |     +--rw client-cert-auth {sshcmn:ssh-x509-certs}?
        |        |     |  +--rw pinned-ca-certs?
        |        |     |  |       ta:pinned-certificates-ref
        |        |     |  +--rw pinned-client-certs?
        |        |     |          ta:pinned-certificates-ref
        |        |     +--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
        |        +--:(tls) {tls-listen}?
        |           +--rw tls
        |              +--rw address            inet:ip-address
        |              +--rw port?              inet:port-number
        |              +--rw server-identity
        |              |  +--rw (local-or-keystore)
        |              |     +--:(local)
        |              |     |  +--rw algorithm
        |              |     |  |       ct:key-algorithm-ref
        |              |     |  +--rw public-key                binary
        |              |     |  +--rw private-key               union
        |              |     |  +--rw cert
        |              |     |  |       ct:end-entity-cert-cms
        |              |     |  +---n certificate-expiration
        |              |     |     +-- expiration-date?
        |              |     |             yang:date-and-time
        |              |     +--:(keystore) {keystore-implemented}?
        |              |        +--rw reference
```

```
        |              |                ks:asymmetric-key-certificate-r\
   ef
        |              +--rw client-auth
        |              |  +--rw pinned-ca-certs?
        |              |  |       ta:pinned-certificates-ref
        |              |  +--rw pinned-client-certs?
        |              |  |       ta:pinned-certificates-ref
        |              |  +--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 endpoints
             |  +--rw endpoint* [name]
             |     +--rw name         string
             |     +--rw (transport)
             |        +--:(ssh) {ssh-call-home}?
             |        |  +--rw ssh
             |        |     +--rw address             inet:host
             |        |     +--rw port?               inet:port-number\

             |        |     +--rw server-identity
             |        |     |  +--rw host-key* [name]
             |        |     |     +--rw name                 string
             |        |     |     +--rw (host-key-type)
             |        |     |        +--:(public-key)
             |        |     |        |  +--rw public-key
             |        |     |        |     +--rw (local-or-keystore)
             |        |     |        |        +--:(local)
             |        |     |        |        |  +--rw algorithm
             |        |     |        |        |  |       ct:key-algori\
   thm-ref
             |        |     |        |        |  +--rw public-key
             |        |     |        |        |  |       binary
             |        |     |        |        |  +--rw private-key
             |        |     |        |        |          union
             |        |     |        |        +--:(keystore)
```

```
             |        |     |        |                 {keystore-imple\
   mented}?
             |        |     |        |           +--rw reference
             |        |     |        |                   ks:asymmetric\
   -key-ref
             |        |     |        +--:(certificate)
             |        |     |           +--rw certificate
             |        |     |                   {sshcmn:ssh-x509-certs\
   }?
             |        |     |              +--rw (local-or-keystore)
             |        |     |                 +--:(local)
             |        |     |                 |  +--rw algorithm
             |        |     |                 |  |       ct:key-algori\
   thm-ref
             |        |     |                 |  +--rw public-key
             |        |     |                 |  |       binary
             |        |     |                 |  +--rw private-key
             |        |     |                 |  |       union
             |        |     |                 |  +--rw cert
             |        |     |                 |  |       ct:end-entity\
   -cert-cms
             |        |     |                 |  +---n certificate-exp\
   iration
             |        |     |                 |     +-- expiration-dat\
   e?
             |        |     |                 |             yang:date-\
   and-time
             |        |     |                 +--:(keystore)
             |        |     |                          {keystore-imple\
   mented}?
             |        |     |                    +--rw reference
             |        |     |                            ks:asymmetric\
   -key-certificate-ref
             |        |     +--rw client-cert-auth
             |        |     |       {sshcmn:ssh-x509-certs}?
             |        |     |  +--rw pinned-ca-certs?
             |        |     |  |       ta:pinned-certificates-ref
             |        |     |  +--rw pinned-client-certs?
             |        |     |          ta:pinned-certificates-ref
             |        |     +--rw transport-params
             |        |             {ssh-server-transport-params-confi\
   g}?
             |        |        +--rw host-key
             |        |        |  +--rw host-key-alg*   identityref
             |        |        +--rw key-exchange
             |        |        |  +--rw key-exchange-alg*   identityre\
   f
             |        |        +--rw encryption
```

```
               |        |        |  +--rw encryption-alg*   identityref
               |        |        +--rw mac
               |        |           +--rw mac-alg*   identityref
               |        +--:(tls) {tls-call-home}?
               |           +--rw tls
               |              +--rw address            inet:host
               |              +--rw port?              inet:port-number
               |              +--rw server-identity
               |              |  +--rw (local-or-keystore)
               |              |     +--:(local)
               |              |     |  +--rw algorithm
               |              |     |  |       ct:key-algorithm-ref
               |              |     |  +--rw public-key
               |              |     |  |       binary
               |              |     |  +--rw private-key
               |              |     |  |       union
               |              |     |  +--rw cert
               |              |     |  |       ct:end-entity-cert-cms
               |              |     |  +---n certificate-expiration
               |              |     |     +-- expiration-date?
               |              |     |             yang:date-and-time
               |              |     +--:(keystore) {keystore-implemented\
   }?
               |              |        +--rw reference
               |              |                ks:asymmetric-key-certifi\
   cate-ref
               |              +--rw client-auth
               |              |  +--rw pinned-ca-certs?
               |              |  |       ta:pinned-certificates-ref
               |              |  +--rw pinned-client-certs?
               |              |  |       ta:pinned-certificates-ref
               |              |  +--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
```

4.2. 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].

[Note: '\' line wrapping for formatting only]

```
<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"\
>

  <!-- endpoints to listen for NETCONF connections on -->
  <listen>
    <endpoint> <!-- listening for SSH connections -->
      <name>netconf/ssh</name>
      <ssh>
        <address>192.0.2.7</address>
        <server-identity>
          <host-key>
            <name>deployment-specific-certificate</name>
            <public-key>
              <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-\
crypto-types">ct:secp521r1</algorithm>
              <private-key>base64encodedvalue==</private-key>
              <public-key>base64encodedvalue==</public-key>
            </public-key>
          </host-key>
        </server-identity>
        <client-cert-auth>
```

```
            <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinne\
   d-ca-certs>
            <pinned-client-certs>explicitly-trusted-client-certs</pinn\
   ed-client-certs>
          </client-cert-auth>
        </ssh>
      </endpoint>
      <endpoint> <!-- listening for TLS sessions -->
        <name>netconf/tls</name>
        <tls>
          <address>192.0.2.7</address>
          <server-identity>
            <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-cryp\
   to-types">ct:secp521r1</algorithm>
            <private-key>base64encodedvalue==</private-key>
            <public-key>base64encodedvalue==</public-key>
            <cert>base64encodedvalue==</cert>
          </server-identity>
          <client-auth>
            <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinne\
   d-ca-certs>
            <pinned-client-certs>explicitly-trusted-client-certs</pinn\
   ed-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 SSH and TLS based NETCONF clients -->
    <call-home>
      <netconf-client> <!-- SSH-based client -->
        <name>config-mgr</name>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
```

```
            <ssh>
              <address>east.config-mgr.example.com</address>
              <server-identity>
                <host-key>
                  <name>deployment-specific-certificate</name>
                  <public-key>
                    <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:i\
   etf-crypto-types">ct:secp521r1</algorithm>
                    <private-key>base64encodedvalue==</private-key>
                    <public-key>base64encodedvalue==</public-key>
                  </public-key>
                </host-key>
              </server-identity>
              <client-cert-auth>
                <pinned-ca-certs>explicitly-trusted-client-ca-certs</p\
   inned-ca-certs>
                <pinned-client-certs>explicitly-trusted-client-certs</\
   pinned-client-certs>
              </client-cert-auth>
            </ssh>
          </endpoint>
          <endpoint>
            <name>west-data-center</name>
            <ssh>
              <address>west.config-mgr.example.com</address>
              <server-identity>
                <host-key>
                  <name>deployment-specific-certificate</name>
                  <public-key>
                    <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:i\
   etf-crypto-types">ct:secp521r1</algorithm>
                    <private-key>base64encodedvalue==</private-key>
                    <public-key>base64encodedvalue==</public-key>
                  </public-key>
                </host-key>
              </server-identity>
              <client-cert-auth>
                <pinned-ca-certs>explicitly-trusted-client-ca-certs</p\
   inned-ca-certs>
                <pinned-client-certs>explicitly-trusted-client-certs</\
   pinned-client-certs>
              </client-cert-auth>
            </ssh>
          </endpoint>
        </endpoints>
        <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>data-collector</name>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <tls>
              <address>east.analytics.example.com</address>
              <server-identity>
                <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-\
   crypto-types">ct:secp521r1</algorithm>
                <private-key>base64encodedvalue==</private-key>
                <public-key>base64encodedvalue==</public-key>
                <cert>base64encodedvalue==</cert>
              </server-identity>
              <client-auth>
                <pinned-ca-certs>explicitly-trusted-client-ca-certs</p\
   inned-ca-certs>
                <pinned-client-certs>explicitly-trusted-client-certs</\
   pinned-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>
          <endpoint>
            <name>west-data-center</name>
            <tls>
              <address>west.analytics.example.com</address>
              <server-identity>
```

```
                <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-\
   crypto-types">ct:secp521r1</algorithm>
                <private-key>base64encodedvalue==</private-key>
                <public-key>base64encodedvalue==</public-key>
                <cert>base64encodedvalue==</cert>
              </server-identity>
              <client-auth>
                <pinned-ca-certs>explicitly-trusted-client-ca-certs</p\
   inned-ca-certs>
                <pinned-client-certs>explicitly-trusted-client-certs</\
   pinned-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>
        </endpoints>
        <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>
```

4.3. YANG Module

This YANG module has normative references to [RFC6242], [RFC6991], [RFC7407], [RFC7589], [RFC8071], [I-D.ietf-netconf-ssh-client-server], and [I-D.ietf-netconf-tls-client-server].

This YANG module imports YANG types from [RFC6991], and YANG groupings from [RFC7407], [I-D.ietf-netconf-ssh-client-server] and [I-D.ietf-netconf-ssh-client-server].

```
<CODE BEGINS> file "ietf-netconf-server@2018-06-04.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 2018-06-04; // stable grouping definitions
    reference
      "RFC YYYY: YANG Groupings for SSH Clients and SSH Servers";
  }

  import ietf-tls-server {
    prefix ts;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC ZZZZ: YANG Groupings for TLS Clients and TLS Servers";
  }

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

  contact
```

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

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

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

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


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

        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 "2018-06-04" {
        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 listen {
         if-feature listen;
         presence "Enables server to listen for TCP connections";
         description "Configures listen behavior";
         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;
           min-elements 1;
           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;
```

```
                    mandatory true;
                    description
                      "The IP address to listen on for incoming
                       connections.  The NETCONF server will listen
                       on all configured interfaces if no value is
                       specified.  INADDR_ANY (0.0.0.0) or INADDR6_ANY
                       (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
                       the server is to listen on all IPv4 or IPv6
                       addresses, respectively.";
                  }
                  leaf port {
                    type inet:port-number;
                    default 830;
                    description
                     "The local port number to listen on.  If no value
                      is specified, the IANA-assigned port value for
                      'netconf-ssh' (830) is used.";
                  }
                  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;
                    mandatory true;
                    description
                      "The IP address to listen on for incoming
                       connections.  The NETCONF server will listen
                       on all configured interfaces if no value is
                       specified.  INADDR_ANY (0.0.0.0) or INADDR6_ANY
                       (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
                       the server is to listen on all IPv4 or IPv6
                       addresses, respectively.";
                  }
                  leaf port {
                    type inet:port-number;
                    default 6513;
                    description
                     "The local port number to listen on.  If no value
                      is specified, the IANA-assigned port value for
                      'netconf-tls' (6513) is used.";
                  }
                  uses ts:tls-server-grouping {
```

```
                    refine "client-auth" {
                      must 'pinned-ca-certs or pinned-client-certs';
                      description
                        "NETCONF/TLS servers MUST validate client
                         certiticates.";
                    }
                    augment "client-auth" {
                      description
                        "Augments in the cert-to-name structure.";
                      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 WWWW: NETCONF over TLS, Section 7";
                      }
                    }
                  }
                }
              }
            }
          }
        }

        container call-home {
          if-feature call-home;
          presence "Enables server to initiate TCP connections";
          description "Configures call-home behavior";
          list netconf-client {
            key name;
            min-elements 1;
            description
              "List of NETCONF clients the NETCONF server is to
               initiate call-home connections to in parallel.";
            leaf name {
              type string;
              description
                "An arbitrary name for the remote NETCONF client.";
            }
            container endpoints {
              description
```

```
              "Container for the list of endpoints.";
            list endpoint {
              key name;
              min-elements 1;
              ordered-by user;
              description
                "A non-empty user-ordered list of endpoints for this
                 NETCONF server to try to connect to in sequence.
                 Defining more than one enables high-availability.";
              leaf name {
                type string;
                description
                  "An arbitrary name for this endpoint.";
              }
              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.";
                    leaf address {
                      type inet:host;
                      mandatory true;
                      description
                       "The IP address or hostname of the endpoint.
                        If a domain name is configured, then the
                        DNS resolution should happen on each usage
                        attempt.  If the the DNS resolution results
                        in multiple IP addresses, the IP addresses
                        will be tried according to local preference
                        order until a connection has been established
                        or until all IP addresses have failed.";
                    }
                    leaf port {
                      type inet:port-number;
                      default 4334;
                      description
                       "The IP port for this endpoint. The NETCONF
                        server will use the IANA-assigned well-known
                        port for 'netconf-ch-ssh' (4334) if no value
                        is specified.";
                    }
                    uses ss:ssh-server-grouping;
                  }
```

```
                  }
                  case tls {
                    if-feature tls-call-home;
                    container tls {
                      description
                        "Specifies TLS-specific call-home transport
                         configuration.";
                      leaf address {
                        type inet:host;
                        mandatory true;
                        description
                         "The IP address or hostname of the endpoint.
                          If a domain name is configured, then the
                          DNS resolution should happen on each usage
                          attempt.  If the the DNS resolution results
                          in multiple IP addresses, the IP addresses
                          will be tried according to local preference
                          order until a connection has been established
                          or until all IP addresses have failed.";
                      }
                      leaf port {
                        type inet:port-number;
                        default 4335;
                        description
                         "The IP port for this endpoint.  The NETCONF
                          server will use the IANA-assigned well-known
                          port for 'netconf-ch-tls' (4335) if no value
                          is specified.";
                      }
                      uses ts:tls-server-grouping {
                        refine "client-auth" {
                         must 'pinned-ca-certs or pinned-client-certs';
                         description
                          "NETCONF/TLS servers MUST validate client
                           certiticates.";
                        }
                        augment "client-auth" {
                          description
                            "Augments in the cert-to-name structure.";
                          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 WWWW: NETCONF over TLS, Section 7";
                          }
                        }
                      }
                    }
                  } // end tls
                } // end choice
              } // end endpoint
            }
            container connection-type {
              description
               "Indicates the kind of connection to use.";
              choice connection-type {
                default persistent-connection;
                description
                  "Selects between available connection types.";
                case persistent-connection {
                  container persistent {
                    presence
                     "Indicates that a persistent connection is to be
                      maintained.";
                    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 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
                     "Indicates that a periodic connection is to be
                      maintained.";
                    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 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 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.   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).";
             }
           }
         }
       }
     }
   }

   <CODE ENDS>
```

5. Design Considerations

Editorial: this section is a hold over from before, previously called "Objectives". It was only written two 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:

5.1. 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.

5.2. 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.

5.3. 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.

5.4. 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.

5.5. 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.

5.6. For Call Home connections

The following objectives only pertain to call home connections.

5.6.1. 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.

5.6.2. 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.

5.6.3. 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.

5.6.4. 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.

5.6.5. 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.

5.6.6. 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.

5.6.7. 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).

6. Security Considerations

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:

/: The entire data trees defined by the modules defined in this draft are 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

7. IANA Considerations

7.1. 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.

7.2. 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
```

8. References

8.1. Normative References

[I-D.ietf-netconf-keystore]
Watsen, K., "YANG Data Model for a "Keystore" Mechanism", draft-ietf-netconf-keystore-04 (work in progress), October 2017.

[I-D.ietf-netconf-ssh-client-server]
Watsen, K. and G. Wu, "YANG Groupings for SSH Clients and SSH Servers", draft-ietf-netconf-ssh-client-server-05 (work in progress), October 2017.

[I-D.ietf-netconf-tls-client-server]
Watsen, K. and G. Wu, "YANG Groupings for TLS Clients and TLS Servers", draft-ietf-netconf-tls-client-server-05 (work in progress), October 2017.

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

[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, <https://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, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc3688>.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <https://www.rfc-editor.org/info/rfc6536>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Change Log

A.1. 00 to 01

- Renamed "keychain" to "keystore".

A.2. 01 to 02

- Added to ietf-netconf-client ability to connected to a cluster of endpoints, including a reconnection-strategy.

- Added to ietf-netconf-client the ability to configure connection-type and also keep-alive strategy.

- Updated both modules to accomodate new groupings in the ssh/tls drafts.

A.3. 02 to 03

- Refined use of tls-client-grouping to add a must statement indicating that the TLS client must specify a client-certificate.

- Changed ’netconf-client’ to be a grouping (not a container).

A.4. 03 to 04

- Added RFC 8174 to Requirements Language Section.

- Replaced refine statement in ietf-netconf-client to add a mandatory true.

- Added refine statement in ietf-netconf-server to add a must statement.

- Now there are containers and groupings, for both the client and server models.

A.5. 04 to 05

- Now tree diagrams reference ietf-netmod-yang-tree-diagrams

- Updated examples to inline key and certificates (no longer a leafref to keystore)

A.6. 05 to 06

- o Fixed change log missing section issue.
- o Updated examples to match latest updates to the crypto-types, trust-anchors, and keystore drafts.
- o Reduced line length of the YANG modules to fit within 69 columns.

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.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
VMware
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
July 2, 2018

NETCONF Support for Event Notifications
draft-ietf-netconf-netconf-event-notifications-10

Abstract

This document provides a NETCONF binding to subscribed notifications and to YANG push.

RFC Editor note: please replace the four references to pre-RFC normative drafts with the actual assigned RFC numbers.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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. Compatibility with RFC-5277's create-subscription 3
4. Mandatory XML, event stream and datastore support 4
5. NETCONF connectivity and the subscription lifecycle 4
5.1. Dynamic Subscriptions 4
5.2. Configured Subscriptions 4
6. Notification Messages 5
7. Dynamic Subscriptions and RPC Error Responses 5
8. YANG module 7
9. IANA Considerations 9
10. Security Considerations 9
11. Acknowledgments 9
12. References 9
12.1. Normative References 10
12.2. Informative References 11
Appendix A. Examples 11
A.1. Event Stream Discovery 11
A.2. Dynamic Subscriptions 12
A.3. Configured Subscriptions 16
A.4. Subscription State Notifications 22
Appendix B. Changes between revisions 24
B.1. v09 to v10 24
B.2. v08 to v09 24
B.3. v07 to v08 24
B.4. v06 to v07 24
B.5. v05 to v06 25
B.6. v03 to v04 25
B.7. v01 to v03 25
B.8. v00 to v01 25
Authors' Addresses 25

1. Introduction

This document provides a binding for events streamed over the NETCONF protocol [RFC6241] as per [I-D.draft-ietf-netconf-subscribed-notifications]. In addition, as [I-D.ietf-netconf-yang-push] is itself built upon [I-D.draft-ietf-netconf-subscribed-notifications], this document

enables a NETCONF client to request and receive updates from a YANG datastore located on a NETCONF server.

2. 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.

The following terms are defined in [I-D.draft-ietf-netconf-subscribed-notifications]: notification message, event stream, publisher, receiver, subscriber, subscription, configured subscription.

3. Compatibility with RFC-5277’s create-subscription

A publisher is allowed to concurrently support configured subscriptions and dynamic subscription RPCs of [I-D.draft-ietf-netconf-subscribed-notifications] at the same time as [RFC5277]’s "create-subscription" RPC. However a single NETCONF transport session cannot support both this specification and a subscription established by [RFC5277]’s "create-subscription" RPC. To protect against any attempts to use a single NETCONF transport session in this way:

- o A solution must reply with the [RFC6241] error "operation-not-supported" if a "create-subscription" RPC is received on a NETCONF session where any other [I-D.draft-ietf-netconf-subscribed-notifications] or [RFC5277] subscription exists.
- o It is a prohibited to send updates or state change notifications for a configured subscription on a NETCONF session where the create-subscription RPC has successfully [RFC5277] created subscription.
- o A "create-subscription" RPC MUST be rejected if any [I-D.draft-ietf-netconf-subscribed-notifications] or [RFC5277]subscription is active across that NETCONF transport session.

If a publisher supports this specification but not subscriptions via [RFC5277], the publisher MUST NOT advertise "urn:ietf:params:netconf:capability:notification:1.0".

4. Mandatory XML, event stream and datastore support

The "encode-xml" feature of [I-D.draft-ietf-netconf-subscribed-notifications] is mandatory to support. This indicates that XML is a valid encoding for RPCs, state change notifications, and subscribed content.

A NETCONF publisher supporting event stream subscription via [I-D.draft-ietf-netconf-subscribed-notifications] MUST support the "NETCONF" event stream identified in that draft.

A NETCONF publisher supporting [I-D.ietf-netconf-yang-push] MUST support the operational state datastore as defined by [RFC8342].

5. NETCONF connectivity and the subscription lifecycle

This section describes how the availability of NETCONF transport impacts the establishment and lifecycle of different types of [I-D.draft-ietf-netconf-subscribed-notifications] subscriptions.

5.1. Dynamic Subscriptions

For a dynamic subscription, if the NETCONF session involved with the "establish-subscription" terminates, the subscription MUST be deleted.

For a dynamic subscription a "modify-subscription", "delete-subscription", or "resynch-subscription" RPC MUST be sent using same the NETCONF session upon which the referenced subscription was established.".

5.2. Configured Subscriptions

When a configured subscription enters the "valid" state, there is no guarantee a usable NETCONF transport session is currently in place with each associated receiver. As a result, the first configured subscription to a specific receiver MUST establish a NETCONF transport session via NETCONF call home [RFC8071], section 4.1. This transport session MUST then be used by additional configured subscriptions targeting that the same receiver. This same receiver is identifiable on the publisher as one which targets the same IP address and port used to establish the existing NETCONF call home connection. This transport session MAY also be used by dynamic subscriptions and/or non-subscription related NETCONF operations originated by the NETCONF client.

The method of identifying the targeted recevier IP address, port, and security credentials are left up to implementers of this

specification. For implementation guidance and a YANG model for this function, please look to [I-D.draft-ietf-netconf-netconf-client-server].

If the call home fails because the publisher receives receiver credentials which are subsequently declined per [RFC8071], Section 4.1, step S5 authentication, then that receiver MUST be placed into the "timeout" state.

If the call home fails to establish for any other reason, the publisher MUST NOT progress the receiver to the "active" state. Additionally, the publisher SHOULD place the receiver into the "timeout" state after a predetermined number of either failed call home attempts or NETCONF sessions remotely terminated by the receiver.

NETCONF transport session connectivity SHOULD be verified as described in [RFC8071], Section 4.1, step S7.

Until NETCONF transport with a receiver has been established, and a "subscription-started" state change notification has been successfully sent for a configured subscription, that subscription’s receiver MUST remain in either the "connecting" or the "timeout" state.

If a NETCONF session is disconnected but the "stop-time" of a subscription being transported over that session has not been reached, the publisher restarts the call home process and return the receiver to the "connecting" state.

6. Notification Messages

Notification messages transported over the NETCONF protocol will use the "notification" message defined within [RFC5277], section 4.

All notification messages MUST use the NETCONF transport session used by the "establish-subscription" RPC.

7. Dynamic Subscriptions and RPC Error Responses

Management of dynamic subscriptions occurs via RPCs as defined in [I-D.ietf-netconf-yang-push] and [I-D.draft-ietf-netconf-subscribed-notifications]. When an RPC error occurs, the NETCONF RPC reply MUST include an "rpc-error" element per [RFC6241] with the error information populated as follows:

- an "error-type" node of "application".
- an "error-tag" node of "operation-failed".

- o an "error-severity" of "error" (this MAY but does not have to be included).
- o an "error-app-tag" node with the value being a string that corresponds to an identity associated with the error, as defined in [I-D.draft-ietf-netconf-subscribed-notifications] section 2.4.6 for general subscriptions, and [I-D.ietf-netconf-yang-push] Appendix A.1, for datastore subscriptions. The identityname to use depends on the RPC for which the error occurred. Viable errors for different RPCs are as follows:

RPC	use base identity
establish-subscription	establish-subscription-error
modify-subscription	modify-subscription-error
delete-subscription	delete-subscription-error
kill-subscription	kill-subscription-error
resynch-subscription	resynch-subscription-error

Each error identity will be inserted as the "error-app-tag" using JSON encoding following the form <modulename>:<identityname>. An example of such as valid encoding would be "ietf-subscribed-notifications:no-such-subscription".

- o In case of error responses to an "establish-subscription" or "modify-subscription" request there is the option of including an "error-info" node. This node may contain XML-encoded data with hints for parameter settings that might lead to successful RPC requests in the future. Following are the yang-data structures which may be returned:

```
establish-subscription returns hints in yang-data structure
---------------------- ------------------------------------
target: event stream   establish-subscription-stream-error-info
target: datastore      establish-subscription-datastore-error-info

modify-subscription    returns hints in yang-data structure
---------------------- ------------------------------------
target: event stream   modify-subscription-stream-error-info
target: datastore      modify-subscription-datastore-error-info
```

The yang-data included within "error-info" SHOULD NOT include the optional leaf "error-reason", as such a leaf would be redundant with information that is already placed within the "error-app-tag".

In case of an rpc error as a result of a "delete-subscription", a "kill-subscription", or a "resynch-subscription" request, no "error-info" needs to be included, as the "subscription-id" is the only RPC input parameter and no hints regarding this RPC input parameters need to be provided.

8. YANG module

This module references [I-D.draft-ietf-netconf-subscribed-notifications].

[note to the RFC Editor - please replace XXXX within this YANG model with the number of this document]

```
<CODE BEGINS>file
"ietf-netconf-subscribed-notifications@2018-04-20.yang"
module ietf-netconf-subscribed-notifications {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-netconf-subscribed-notifications";

  prefix nsn;

  import ietf-subscribed-notifications {
    prefix sn;
  }

  organization "IETF NETCONF (Network Configuration) Working Group";
  contact
    "WG Web:   <http:/tools.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>

     Editor:   Eric Voit
```

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

     Editor:   Alexander Clemm
               <mailto:ludwig@clemm.org>

     Editor:   Alberto Gonzalez Prieto
               <mailto:agonzalezpri@vmware.com>

     Editor:   Ambika Prasad Tripathy
               <mailto:ambtripa@cisco.com>

     Editor:   Einar Nilsen-Nygaard
               <mailto:einarnn@cisco.com>";


  description
    "Defines NETCONF as a supported transport for subscribed event
    notifications.

    Copyright (c) 2018 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
    (https://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 2018-04-20 {
    description
      "Initial version";
    reference
    "RFC XXXX: NETCONF Support for Event Notifications";
  }

  identity netconf {
    base sn:transport;
    base sn:inline-address;
    description
      "NETCONF is used as a transport for notification messages and
       state change notifications.";
  }
}
<CODE ENDS>
```

9. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-subscribed-notifications
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-netconf-subscribed-notifications
Namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-subscribed-notifications
Prefix: nsn
Reference: RFC XXXX: NETCONF Support for Event Notifications

10. Security Considerations

Notification messages (including state change notifications) are never sent before the NETCONF capabilities exchange has completed.

If a malicious or buggy NETCONF subscriber 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 underlying NETCONF session. The publisher MAY also suspend or terminate a subset of the active subscriptions on that NETCONF session.

This draft has a YANG module which consists of a single identity. As a result additional security concerns beyond those of the imported modules are not introduced.

11. 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, Martin Bjorklund, Mahesh Jethanandani, Kent Watsen, and Guangying Zheng.

12. References

12.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Customized Subscriptions to a Publisher's Event Streams", draft-ietf-netconf-subscribed-notifications-14 (work in progress), July 2018.

[I-D.ietf-netconf-yang-push]
Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", June 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6241>.

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

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

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

12.2. Informative References

[I-D.draft-ietf-netconf-netconf-client-server]
Watsen, K. and G. Wu, "NETCONF Client and Server Models", draft-ietf-netconf-netconf-client-server-06 (work in progress), June 2018.

Appendix A. Examples

This section is non-normative.

A.1. Event Stream Discovery

As defined in [I-D.draft-ietf-netconf-subscribed-notifications] 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 publisher using the "get" operation against the top-level container "/streams" defined in [I-D.draft-ietf-netconf-subscribed-notifications] Section 3.1.

The following example illustrates the retrieval of the list of available event streams:

```
<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:yang:ietf-subscribed-notifications"/>
    </filter>
  </get>
</rpc>
```

Figure 1: Get streams request

After such a request, the NETCONF publisher returns a list of event streams available, as well as additional information which might exist in the container.

A.2. Dynamic Subscriptions

A.2.1. Establishing Dynamic Subscriptions

The following figure shows two successful "establish-subscription" RPC requests as per [I-D.draft-ietf-netconf-subscribed-notifications]. The first request is given a subscription identifier of 22, the second, an identifier of 23.

```
+------------+                 +-----------+
| Subscriber |                 | Publisher |
+------------+                 +-----------+
      |                              |
      |    Capability Exchange       |
      |<----------------------------->|
      |                              |
      |                              |
      |    establish-subscription    |
      |----------------------------->|  (a)
      | RPC Reply: OK, id = 22       |
      |<-----------------------------|  (b)
      |                              |
      | notification message (for 22)|
      |<-----------------------------|
      |                              |
      |                              |
      |    establish-subscription    |
      |----------------------------->|
      | notification message (for 22)|
      |<-----------------------------|
      | RPC Reply: OK, id = 23       |
      |<-----------------------------|
      |                              |
      |                              |
      | notification message (for 22)|
      |<-----------------------------|
      | notification message (for 23)|
      |<-----------------------------|
      |                              |
```

Figure 2: Multiple subscriptions over a NETCONF session

To provide examples of the information being transported, example messages for interactions (a) and (b) in Figure 2 are detailed below:

```
<rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <establish-subscription
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <stream>NETCONF</stream>
    <stream-xpath-filter xmlns:ex="http://example.com/events">
      /ex:foo/
    </stream-xpath-filter>
    <dscp>10</dscp>
  </establish-subscription>
</rpc>
```

Figure 3: establish-subscription request (a)

As NETCONF publisher was able to fully satisfy the request (a), the publisher sends the subscription identifier of the accepted subscription within message (b):

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <identifier
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    22
  </identifier>
</rpc-reply>
```

Figure 4: establish-subscription success (b)

If the NETCONF publisher had not been able to fully satisfy the request, or subscriber has no authorization to establish the subscription, the publisher would have sent an RPC error response. For instance, if the "dscp" value of 10 asserted by the subscriber in Figure 3 proved unacceptable, the publisher may have returned:

```
<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
   <error-type>application</error-type>
   <error-tag>operation-failed</error-tag>
   <error-severity>error</error-severity>
   <error-app-tag>
     ietf-subscribed-notifications:dscp-unavailable
   </error-app-tag>
  </rpc-error>
</rpc-reply>
```

Figure 5: an unsuccessful establish subscription

The subscriber can use this information in future attempts to establish a subscription.

A.2.2. Modifying Dynamic Subscriptions

An existing subscription may be modified. The following exchange shows a negotiation of such a modification via several exchanges between a subscriber and a publisher. This negotiation consists of a failed RPC modification request/response, followed by a successful one.

```
+------------+                 +-----------+
| Subscriber |                 | Publisher |
+------------+                 +-----------+
      |                              |
      | notification message (for 23)|
      |<-----------------------------|
      |                              |
      | modify-subscription (id = 23)|
      |----------------------------->|  (c)
      | RPC error (with hint)        |
      |<-----------------------------|  (d)
      |                              |
      | modify-subscription (id = 23)|
      |----------------------------->|
      | RPC Reply: OK                |
      |<-----------------------------|
      |                              |
      | notification message (for 23)|
      |<-----------------------------|
      |                              |
```

Figure 6: Interaction model for successful subscription modification

If the subscription being modified in Figure 6 is a datastore subscription as per [I-D.ietf-netconf-yang-push], the modification request made in (c) may look like that shown in Figure 7. As can be seen, the modifications being attempted are the application of a new xpath filter as well as the setting of a new periodic time interval.

```
<rpc message-id="303"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <modify-subscription
       xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <identifier>23</identifier>
    <yp:datastore-xpath-filter xmlns="http://example.com/datastore">
        /interfaces-state/interface/oper-status
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
  </modify-subscription>
</rpc>
```

Figure 7: Subscription modification request (c)

If the NETCONF publisher can satisfy both changes, the publisher sends a positive result for the RPC. If the NETCONF publisher cannot satisfy either of the proposed changes, the publisher sends an RPC error response (d). The following is an example RPC error response for (d) which includes a hint. This hint is an alternative time period value which might have resulted in a successful modification:

```
<rpc-reply message-id="303"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>
        ietf-yang-push:period-unsupported
    </error-app-tag>
    <error-info
       xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
       <modify-subscription-datastore-error-info>
         <period-hint>
             3000
         </period-hint>
       </modify-subscription-datastore-error-info>
    </error-info>
  </rpc-error>
</rpc-reply>
```

Figure 8: Modify subscription failure with Hint (d)

A.2.3. Deleting Dynamic Subscriptions

The following demonstrates deleting a subscription. This subscription may have been to either a stream or a datastore.

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <delete-subscription
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <identifier>22</identifier>
  </delete-subscription>
</rpc>
```

Figure 9: Delete subscription

If the NETCONF publisher can satisfy the request, the publisher replies with success to the RPC request.

If the NETCONF publisher cannot satisfy the request, the publisher sends an error-rpc element indicating the modification didn’t work. Figure 10 shows a valid response for existing valid subscription identifier, but that subscription identifier was created on a different NETCONF transport session:

```
<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>
        ietf-subscribed-notifications:no-such-subscription
    </error-app-tag>
  </rpc-error>
</rpc-reply>
```

Figure 10: Unsuccessful delete subscription

A.3. Configured Subscriptions

Configured subscriptions may be established, modified, and deleted using configuration operations against the top-level subtree of [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push].

In this section, we present examples of how to manage the configuration subscriptions using a NETCONF client.

A.3.1. Creating Configured Subscriptions

For subscription creation, a NETCONF client may send:

```
<rpc message-id="201" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <default-operation>none</default-operation>
    <config>
      <subscriptions
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
        <subscription>
          <identifier>1922</identifier>
          <transport xmlns:nsn=
    "urn:ietf:params:xml:ns:yang:ietf-netconf-subscribed-notifications">
            nsn:netconf
          <transport>
          <stream>NETCONF</stream>
          <receivers>
            <receiver>
              <name>receiver1</name>
            </receiver>
          </receivers>
        </subscription>
      </subscriptions>
    </config>
  </edit-config>
</rpc>
```

Figure 11: Create a configured subscription

If the request is accepted, the publisher will indicate this. If the request is not accepted because the publisher cannot serve it, no configuration is changed. In this case the publisher may reply:

```
<rpc-reply message-id="201"
  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 12: Response to a failed configured subscription establishment

After a subscription has been created, NETCONF connectivity to each receiver will be established if it does not already exist. This will be accomplished through the association on the publisher with the networking parameters needed to establish connectivity with each configured receiver. These parameters will be used as needed by NETCONF call home [RFC8071].

The following figure shows the interaction model for the successful creation of a configured subscription.

```
    +----------+                 +-----------+     +---------+
    |Config Ops|                 | Publisher |     |Receiver1|
    +----------+                 +-----------+     +---------+
         |                            |                |
         |    Capability Exchange     |                |
         |<-------------------------->|                |
         |                            |                |
         |                            |                |
         |        Edit-config         |                |
         |--------------------------->|                |
         |       RPC Reply: OK        |                |
         |<---------------------------|                |
         |                            |   Call Home    |
         |                            |<-------------->|
         |                            |                |
         |                            |  subscription- |
         |                            |  started       |
         |                            |--------------->|
         |                            |                |
         |                            |  notification  |
         |                            |  message       |
         |                            |--------------->|
```

Figure 13: Interaction model for configured subscription establishment

A.3.2. Modifying Configured Subscriptions

Configured subscriptions can be modified using configuration operations against the top-level container "/subscriptions".

For example, the subscription established in the previous section could be modified as follows, here a adding a second receiver:

```
<rpc message-id="202" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <subscriptions
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
        <subscription>
          <identifier>
            1922
          </identifier>
          <receivers>
            <receiver>
              <name>receiver2</name>
            </receiver>
          </receivers>
        </subscription>
      </subscriptions>
    </config>
  </edit-config>
</rpc>
```

Figure 14: Modify configured subscription

If the request is accepted, the publisher will indicate success. The result is that the interaction model described in Figure 13 may be extended as follows.

```
 +----------+                 +-----------+     +---------+  +---------+
 |Config Ops|                 | Publisher |     |Receiver1|  |Receiver2|
 +----------+                 +-----------+     +---------+  +---------+
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |        Edit-config         |                |            |
       |--------------------------->|                |            |
       |       RPC Reply: OK        |                |            |
       |<---------------------------|                |            |
       |                            |  subscription- |            |
       |                            |  started       |            |
       |                            |----------------------------->|
       |                            |                |            |
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |                            |----------------------------->|
       |                            |                |            |
```

Figure 15: Interaction model for configured subscription modification

Note in the above that in the specific example above, modifying a configured subscription actually resulted in "subscription-started" notification. And because of an existing NETCONF session, no additional call home was needed. Also note that if the edit of the configuration had impacted the filter, a separate modify-subscription would have been required for the original receiver.

A.3.3. Deleting Configured Subscriptions

Configured subscriptions can be deleted using configuration operations against the top-level container "/subscriptions". Deleting the subscription above would result in the following flow impacting all active receivers.

```
 +----------+                 +-----------+     +---------+  +---------+
 |Config Ops|                 | Publisher |     |Receiver1|  |Receiver2|
 +----------+                 +-----------+     +---------+  +---------+
       |                            |                |            |
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |                            |----------------------------->|
       |                            |                |            |
       |        Edit-config         |                |            |
       |--------------------------->|                |            |
       |       RPC Reply: OK        |                |            |
       |<---------------------------|                |            |
       |                            |  subscription- |            |
       |                            |  terminated    |            |
       |                            |--------------->|            |
       |                            |----------------------------->|
       |                            |                |            |
```

Figure 16: Interaction model for configured subscription deletion

A.4. Subscription State Notifications

A publisher will send subscription state notifications according to the definitions within [I-D.draft-ietf-netconf-subscribed-notifications]).

A.4.1. subscription-started and subscription-modified

A "subscription-started" over NETCONF encoded in XML would look like:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-started
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <identifier>39</identifier>
    <transport xmlns:nsn=
    "urn:ietf:params:xml:ns:yang:ietf-netconf-subscribed-notifications">
      nsn:netconf
    <transport>
    <stream-xpath-filter xmlns:ex="http://example.com/events">
      /ex:foo
    </stream-xpath-filter>
    <stream>NETCONF</stream>
  </subscription-started>
</notification>
```

Figure 17: subscription-started subscription state notification

The "subscription-modified" is identical to Figure 17, with just the word "started" being replaced by "modified".

A.4.2. subscription-completed, subscription-resumed, and replay-complete

A "subscription-completed" would look like:

```
  <notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <subscription-completed
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
      <identifier>39</identifier>
    </subscription-completed>
  </notification>
```

Figure 18: subscription-completed notification in XML

The "subscription-resumed" and "replay-complete" are virtually identical, with "subscription-completed" simply being replaced by "subscription-resumed" and "replay-complete".

A.4.3. subscription-terminated and subscription-suspended

A "subscription-terminated" would look like:

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <subscription-terminated
    xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <identifier>39</identifier>
    <error-id>
       suspension-timeout
    </error-id>
  </subscription-terminated>
</notification>
```

Figure 19: subscription-terminated subscription state notification

The "subscription-suspended" is virtually identical, with "subscription-terminated" simply being replaced by "subscription-suspended".

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

B.1. v09 to v10

- o Tweaks to examples and text.
- o Downshifted state names.
- o Removed address from examples.

B.2. v08 to v09

- o Tweaks based on Kent’s comments.
- o Updated examples in Appendix A. And updates to some object names based on changes in the subscribed-notifications draft.
- o Added a YANG model for the NETCONF identity.

B.3. v07 to v08

- o Tweaks and clarification on :interleave.

B.4. v06 to v07

- o XML encoding and operational datastore mandatory.
- o Error mechanisms and examples updated.

B.5. v05 to v06

- o Moved examples to appendices
- o All examples rewritten based on namespace learnings
- o Normative text consolidated in front
- o Removed all mention of JSON
- o Call home process detailed
- o Note: this is a major revision attempting to cover those comments received from two week review.

B.6. 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.7. v01 to v03

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

B.8. 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

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
Internet-Draft
Intended status: Standards Track
Expires: August 24, 2018

E. Voit
Cisco Systems
H. Birkholz
Fraunhofer SIT
A. Bierman
YumaWorks
A. Clemm
Huawei
T. Jenkins
Cisco Systems
February 20, 2018

Notification Message Headers and Bundles
draft-ietf-netconf-notification-messages-03

Abstract

This document defines a new notification message format, using yang-data. Included are:

- o a new notification mechanism and encoding to replace the one way operation of RFC-5277

- o a set of common, transport agnostic message header objects.

- o how to bundle multiple event records into a single notification message.

- o how to ensure these new capabilities are only used with capable receivers.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 24, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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. Encapsulation of Header Objects in Messages 4
4.1. One Notification per Message 5
4.2. Many Notifications per Message 5
5. Configuration of Headers 8
6. Discovering Receiver Support 9
7. YANG Module 10
8. Backwards Compatibility 18
9. Security Considerations 18
10. Acknowledgments 19
11. References 19
11.1. Normative References 19
11.2. Informative References 19
Appendix A. Changes between revisions 20
Appendix B. Issues being worked 21
Appendix C. Subscription Specific Headers 21
Appendix D. Implications to Existing RFCs 22
D.1. Implications to RFC-7950 22
D.2. Implications to RFC-8040 22
Authors' Addresses 22

1. Introduction

Mechanisms to support subscription to event notifications and yang datastore push are being defined in [I-D.draft-ietf-netconf-subscribed-notifications] and [I-D.ietf-netconf-yang-push]. Work on those documents has shown that notifications described in [RFC7950] section 7.16 could benefit from transport independent headers. Communicating the following

information to receiving applications can be done without explicit linkage to an underlying transport protocol:

- o the time information was generated
- o the time the information was placed in a message and queued for transport
- o a signature to verify authenticity
- o the process generating the information
- o an originating request correlation
- o an ability to bundle information records into one 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 structures 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", "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.

The definition of notification is in RFC 7950 [RFC7950]. Publisher, receiver, and subscription are defined in [I-D.draft-ietf-netconf-subscribed-notifications].

3. Header Objects

There are a number of transport independent headers which should have common definition. These include:

- o subscription-id: provides a reference into the reason the publisher believed the receiver wishes to be notified of this specific information.
- o notification-time: the time an event, datastore update, or other item is recognized and recorded within the publisher.
- o notification-id: Identifies the name of the notification, per the YANG notification statement. May also provide the name of a yang-

data statement (whether transporting other types of messages is in scope is tbd).

- o observation-domain-id: identifies the publisher process which discovered and recorded the event notification. (note: look to reuse the domains set up with IPFIX.)

- o message-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 message-id: for a specific message generator, this identifies a message which includes one or more event records. The message-id increments by one with sequential messages.

- o message-generator-id: identifier for the process which created the message. This allows disambiguation of an information source, such as the identification of different line cards sending the messages. Used in conjunction with previous-message-id, this can help find drops and duplications when messages are coming from multiple sources on a device. If there is a message-generator-id in the header, then the previous-message-id MUST be the message-id from the last time that message-generator-id was sent.

4. Encapsulation of Header Objects in Messages

A specific set of well-known objects are of potential use to networking layers prior 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 are these well-known header objects. These objects are identified within a dedicated header subtree which leads off a particular transportable message. This allows header objects to be easily be decoupled, stripped, and processed separately.

There are two types of transportable messages: one format is used when there is one notification being encapsulated, and another format used when there are many notifications being bundled into one message.

A receiver which supporting this document MUST be able to handle receipt of either type of message from an publisher. It is possible

that changes between message types can occur without any prior indication.

4.1. One Notification per Message

This section will be re-instated if NETCONF WG members are not comfortable with the efficiency of the solution which can encode many notifications per message described below.

4.2. Many Notifications per Message

While possible in some scenarios, it often inefficient to marshal and transport every notification independently. Instead, scale and processing speed can be improved by placing multiple notifications into one transportable bundle.

The format of this bundle appears in the yata-data tree below, and is more completely defined in the yang module. There are three parts of this bundle:

- o a message header describing the marshaling, including information such as when the marshaling occurred

- o a list of encapsulated information

- o an optional message footer for whole-message signing and message-generator integrity verification.

Within the list of encapsulated notifications, there are also three parts:

- o a notification header defining what is in an encapsulated notification

- o the actual notification itself

- o an optional notification footer for individual notification signing and observation-domain integrity verification.

```
yang-data message
   +--ro message!
      +--ro message-header
      |  +--ro message-time            yang:date-and-time
      |  +--ro message-id?             uint32
      |  +--ro message-generator-id?   string
      |  +--ro notification-count?     uint16
      +--ro notifications*
      |  +--ro notification-header
      |  |  +--ro notification-time         yang:date-and-time
      |  |  +--ro yang-module?              yang:yang-identifier
      |  |  +--ro yang-notification-name?   notification-type
      |  |  +--ro subscription-id*          uint32
      |  |  +--ro notification-id?          uint32
      |  |  +--ro observation-domain-id?    string
      |  +--ro notification-contents?
      |  +--ro notification-footer!
      |     +--ro signature-algorithm    string
      |     +--ro signature-value        string
      |     +--ro integrity-evidence?    string
      +--ro message-footer!
         +--ro signature-algorithm    string
         +--ro signature-value        string
         +--ro integrity-evidence?    string
```

An XML instance of a message might look like:

```
<yang-data bundled-message
  xmlns="urn:ietf:params:xml:ns:yang:ietf-notification-messages:1.0">
  <message-header>
    <message-time>
          2017-02-14T00:00:05Z
    </message-time>
    <message-id>
          456
    </message-id>
    <notification-count>
          2
    </notification-count>
  </message-header>
  <notifications>
    <notification>
      <notification-header>
        <notification-time>
            2017-02-14T00:00:02Z
        </notification-time>
        <subscription-id>
            823472
        </subscription-id>
        <yang-module>
            ietf-yang-push
        </yang-module>
        <yang-notification-name>
            push-change-update
        </yang-notification-name>
      </notification-header>
      <notification-contents>
        <push-change-update xmlns=
          "urn:ietf:params:xml:ns:yang:ietf-yang-push:1.0">
          <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-contents>
    </notification>
    <notification>
          ...(record #2)...
    </notification>
  </notifications>
</yang-data>
```

5. Configuration of Headers

A publisher MUST select the set of headers to use within any particular message. The two mandatory headers which MUST always be applied are 'message-time' and 'subscription-id'

Beyond these two mandatory headers, additional headers MAY be included. Configuration of what these optional headers should be can come from the following sources:

1. Publisher wide default headers for all notifications. These are included if an optional header is inserted into 'additional-headers' leaf-list shown in the yang tree below.

2. More notification specific headers may also be desired. If new headers are needed for a specific type of YANG notification, these can be populated through 'additional-notification-headers' leaf-list.

3. An application process may also identify common headers to use when transporting notifications for a specific subscription. How these are identified to a publisher is out-of-scope.

The set of headers used for any particular message is the superset of headers for the items listed above.

The YANG tree showing elements of configuration is depicted in the following figure.

```
module: ietf-notification-messages
    +--rw additional-default-headers {publisher}?
       +--rw additional-headers*                        optional-header
       +--rw yang-notification-specific-default*
          |                          [yang-module yang-notification-name]
          +--rw yang-module                          yang:yang-identifier
          +--rw yang-notification-name                notification-type
          +--rw additional-notification-headers*
                                              optional-notification-header
```

Configuration Model structure

Of note in this tree is the optional feature of 'publisher'. This feature indicates an ability to send notifications. A publisher supporting this specification MUST also be able to parse any messages received as defined in this document.

6. Discovering Receiver Support

We need capability exchange from the receiver to the publisher at transport session initiation to indicate support for this specification.

For all types of transport connections, if the receiver indicates support for this specification, then it MAY be used. In addition, [RFC5277] one-way notifications MUST NOT be used if the receiver indicates support for this specification to a publisher which also supports it.

Where NETCONF transport is used, advertising this specification's namespace during an earlier client capabilities discovery phase MAY be used to indicate support for this specification:

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:xml:ns:yang:ietf-notification-messages:1.0
    </capability>
  </capabilities>
  <session-id>4</session-id>
</hello>
```

NOTE: It is understood that even though it is allowed in [RFC6241] section 8.1, robust NETCONF client driven capabilities exchange is not something which is common in implementation. Therefore reviewers are asked to submit alternative proposals to the mailing list.

For RESTCONF, a mechanism for capability discovery is TBD. Proposals are also welcome here.

The mechanism described above assumes that a capability discovery phase happens before a subscription is started. This is not always the case. As an example, consider HTTP2 configured subscriptions from section 3.1.3 of [I-D.draft-ietf-netconf-restconf-notif], there is no guarantee that a capability exchange has taken place before the updates are pushed. A solution for this could be that a receiver would reply "ok" and reply with the client capabilities as part of the POST. (Or just use a different HTTP status code like 202 instead of 200 'ok'). As such a requirement creates a new dependency for [I-D.draft-ietf-netconf-restconf-notif] upon this specification, more discussion is required to decide if this is a viable solution.

7. YANG Module

```
<CODE BEGINS> file "ietf-notification-messages@2018-01-31.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-restconf   { prefix rc;   }


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

     Editor:   Eric Voit
               <mailto:evoit@cisco.com>

     Editor:   Henk Birkholz
               <mailto:henk.birkholz@sit.fraunhofer.de>

     Editor:   Alexander Clemm
               <mailto:ludwig@clemm.org>

     Editor:   Andy Bierman
               <mailto:andy@yumaworks.com>

     Editor:   Tim Jenkins
               <mailto:timjenki@cisco.com>";

  description
    "This module contains conceptual YANG specifications for yang-data
    messages carrying notifications with well known header objects.";

  revision 2018-01-31 {
    description
      "Initial version.";

    reference
      "draft-ietf-netconf-notification-messages-03";
  }

 /*
  * FEATURES
```

```
  */

  feature publisher {
    description
      "This feature indicates that support for both publisher and
      receiver of messages complying to the specification.";
  }

  /*
   * IDENTITIES
   */

   /* Identities for common headers */

  identity common-header {
    description
      "A well known header which can be included somewhere within a
      message.";
  }

  identity message-time {
    base common-header;
    description
      "Header information consisting of time the message headers were
      placed generated prior to being sent to transport";
  }

  identity subscription-id {
    base common-header;
    description
      "Header information consisting of the identifier of the
      subscription associated with the notification being
      encapsulated.";
  }

  identity notification-count {
    base common-header;
    description
      "Header information consisting of the quantity of notifications in
      a bundled-message for a specific receiver.";
  }

  identity optional-header {
    base common-header;
    description
      "A well known header which an application may choose to include
      within a message.";
  }
```

```
  identity message-id {
    base optional-header;
    description
      "Header information that identifies a message to a specific
      receiver";
  }

  identity message-generator-id {
    base optional-header;
    description
      "Header information consisting of an identifier for a software
      entity which created the message (e.g., linecard 1).";
  }

  identity message-signature {
    base optional-header;
    description
      "Identifies two elements of header information consisting of a
      signature and the signtature type for the contents of a message.
      Signatures can be useful for originating applications to
      verify record contents even when shipping over unsecure
      transport.";
  }

  identity message-integrity-evidence {
    base optional-header;
    description
      "Header information consisting of the information which backs up
      the assertions made as to the validity of the information
      provided within the message.";
  }


  identity optional-notification-header {
    base optional-header;
    description
      "A well known header which an application may choose to include
      within a message.";
  }

  identity notification-time  {
    base optional-notification-header;
    description
      "Header information consisting of the time an originating process
      created the notification.";
  }

  identity notification-id {
```

```
    base optional-notification-header;
    description
      "Header information consisting of an identifier for an instance
      of a notification egressing a publisher. ";
  }

  identity observation-domain-id {
    base optional-notification-header;
    description
      "Header information identifying the software entity which created
      the notification (e.g., process id).";
  }

  identity notification-signature {
    base optional-notification-header;
    description
      "Header information consisting of the information which backs up
      the assertions made as to the validity of the information
      provided within the notification.";
  }

  identity notification-integrity-evidence {
    base optional-notification-header;
    description
      "Header information consisting of the information which backs up
      the assertions made as to the validity of the information
      provided within the notification.";
  }



  /*
   * TYPEDEFs
   */

  typedef optional-header {
    type identityref {
      base optional-header;
    }
    description
      "Type of header object which may be included somewhere within a
      message.";
  }

  typedef optional-notification-header {
    type identityref {
      base optional-notification-header;
    }
```

```
    description
      "Type of header object which may be included somewhere within a
      message.";
  }

  typedef notification-type {
    type string {
      pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
    }
    description
      "The name of a notification within a YANG module.";
    reference
      "RFC-7950 Section 7.16";
  }

  /*
   * GROUPINGS
   */

  grouping message-header {
    description
      "Header information included with a message.";
    leaf message-time {
      type yang:date-and-time;
      mandatory true;
      description
        "time the message was generated prior to being sent to
        transport.";
    }
    leaf message-id {
      type uint32;
      description
        "Id for a message going to a receiver from a message
        generator.  The id will increment by one with each message sent
        from a particular message generator, allowing the message-id
        to be used as a sequence number.";
    }
    leaf message-generator-id {
      type string;
      description
        "Software entity which created the message (e.g., linecard 1).
         The combination of message-id and message-generator-id must be
         unique until reset or a roll-over occurs.";
    }
    leaf notification-count {
      type uint16;
      description
        "Quantity of notification records in a bundled-message
```

```
        specific receiver.";
    }
  }

  grouping notification-within-a-module {
    description
      "A location of a notification within a yang model.";
    leaf yang-module {
      type yang:yang-identifier;
      description
        "Name of the YANG module supported by the publisher.";
    }
    leaf yang-notification-name {
      type notification-type;
       description
         "The name of a notification likely from a YANG module.  Note
         that this object should be in the notification contents, so a
         debate is needed whether this is redundant.";
    }
  }

  grouping notification-header {
    description
      "Common informational objects which might help a receiver
      interpret the meaning, details, or importance of a notification.";
    leaf notification-time {
      type yang:date-and-time;
      mandatory true;
      description
        "Time the system recognized the occurrence of an event.";
    }
    uses notification-within-a-module;
    leaf-list subscription-id {
      type uint32;
      description
        "Id of the subscription which led to the notification being
        generated.";
    }
    leaf notification-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).";
```

```
    }
  }

  grouping security-footer {
    description
      "Reusable grouping for common objects which apply to the the
      signing of notifications or messages.";
    leaf signature-algorithm {
      type string;
      mandatory true;
      description
        "The technology with which an originator signed of some
        delineated contents.";
    }
    leaf signature-value {
      type string;
      mandatory true;
      description
        "Any originator signing of the contents of a header and
        content.  This is useful for verifying contents even when
        shipping over unsecure transport.";
    }
    leaf integrity-evidence {
      type string;
      description
        "This mechanism allows a verifier to ensure that the use of the
        private key, represented by the corresponding public key
        certificate, was performed with a TCG compliant TPM
        environment.  This evidence is never included in within any
        signature.";
      reference
        "TCG Infrastructure Workgroup, Subject Key Attestation Evidence
        Extension, Specification Version 1.0, Revision 7.";
    }
  }


  /*
   * YANG-DATA messages for receivers
   */


   rc:yang-data message {
    container message {
      presence
        "Indicates attempt to communicate notifications to a receiver.";
      description
        "Message to a receiver containing one or more notifications";
```

```
      container message-header {
        description
          "Header info for messages.";
        uses message-header;
      }
      list notifications {
        description
          "Set of notifications to a receiver.";
        container notification-header {
          description
            "Header info for a notification.";
          uses notification-header;
        }
        anydata notification-contents {
          description
            "Encapsulates objects following YANG's notification-stmt
            grammar of RFC-7950 section 14.  Within are the notified
            objects the publisher actually generated in order to be
            passed to a receiver after all filtering has completed.";
        }
        container notification-footer {
          presence
            "Indicates attempt to secure a notification.";
          description
            "Signature and evidence for messages.";
          uses security-footer;
        }
      }
      container message-footer {
        presence
          "Indicates attempt to secure the entire message.";
        description
          "Signature and evidence for messages.";
        uses security-footer;
      }
    }
  }

  /*
   * DATA-NODES
   */

  container additional-default-headers {
    if-feature "publisher";
    description
      "This container maintains a list of which additional notifications
      should use which optional headers if the receiver supports this
      specification.";
```

```
    leaf-list additional-headers {
      type optional-header;
      description
        "This list contains the identities of the optional header types
        which are to be included within each message from this
        publisher.";
    }
    list yang-notification-specific-default {
      key "yang-module yang-notification-name";
      description
        "For any included YANG notifications, this list provides
        additional optional headers which should be placed within the
        container notification-header if the receiver supports this
        specification.  This list incrementally adds to any headers
        indicated within the leaf-list 'additional-headers'.";
      uses notification-within-a-module;
      leaf-list additional-notification-headers {
         type optional-notification-header;
         description
           "The set of additional default headers which will be sent
           for a specific YANG notification.";
      }
    }
  }
}

<CODE ENDS>
```

8. Backwards Compatibility

With this specification, there is no change to YANG's 'notification' statement

Legacy clients are unaffected, and existing users of [RFC5277], [RFC7950], and [RFC8040] are free to use current behaviors until all involved device support this specification.

9. Security Considerations

Certain headers might be computationally complex for a publisher to deliver. Signatures or encryption are two examples of this. It MUST be possible to suspend or terminate a subscription due to lack of resources based on this reason.

Decisions on whether to bundle or not to a receiver are fully under the purview of the Publisher. A receiver could slow delivery to existing subscriptions by creating new ones. (Which would result in the publisher going into a bundling mode.)

10. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Martin Bjorklund, Einar Nilsen-Nygaard, and Kent Watsen.

11. References

11.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-06 (work in progress), January 2018.

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

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://www.rfc-editor.org/info/rfc5277>.

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

11.2. Informative References

[I-D.draft-ietf-netconf-restconf-notif]
Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and HTTP transport for event notifications", January 2018, <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, "YANG Datastore Subscription", December 2017, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

[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, <https://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, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

Appendix A. Changes between revisions

(To be removed by RFC editor prior to publication)

v02 - v03

- o Removed the option for an unbundled message. This might be re-added later for transport efficiency if desired by the WG

- o New message structure driven by the desire to put the signature information at the end.

v01 - v02

- o Fixed the yang-data encapsulation container issue

- o Updated object definitions to point to RFC-7950 definitions

- o Added headers for module and notification-type.

v00 - v01

- o Alternative to 5277 one-way notification added

- o Storage of default headers by notification type

- o Backwards compatibility

- o Capability discovery

- o Move to yang-data

- o Removed dscp and record-type as common headers. (Record type can be determined by the namespace of the record contents. Dscp is useful where applications need internal communications within a Publisher, but it is unclear as to whether this use case need be exposed to a receiver.

Appendix B. Issues being worked

(To be removed by RFC editor prior to publication)

Is this capability just for notifications, or is it for any yang-data element too?

A complete JSON document is supposed to be sent as part of Media Type "application/yang-data+json". As we are sending separate notifications after each other, we need to choose whether we start with some extra encapsulation for the very first message pushed, or if we want a new Media Type for streaming updates.

Improved discovery mechanisms for NETCONF

Should we defer support for HTTP2 configured subscriptions until this draft is available? Without capabilities exchange, it might just be easier to wait. In addition, JSON encoding still needs a notification type which is not exising or represented in referenceable in existing yang-models.

Need to ensure the proper references exist to a notification definition driven by RFC-7950 which is acceptable to other eventual users of this specification.

We need to link to Andy Bierman’s anydata extensibility draft for informational purposes. This is under a WG adoption call.

Appendix C. Subscription Specific Headers

(To be removed by RFC editor prior to publication)

This section discusses a future functional addition which could leverage this draft. It is included for informational purposes only.

A dynamic subscriber might want to mandate that certain headers be used for push updates from a publisher. Some examples of this include a subscriber requesting to:

- establish this subscription, but just if transport messages containing the pushed data will be encrypted,

- establish this subscription, but only if you can attest to the information being delivered in requested notification records, or

- provide a sequence-id for all messages to this receiver (in order to check for loss).

Providing this type of functionality would necessitate a new revision of the [I-D.draft-ietf-netconf-subscribed-notifications]'s RPCs and state change notifications. Subscription specific header information would overwrite the default headers identified in this document.

Appendix D. Implications to Existing RFCs

(To be removed by RFC editor prior to publication)

YANG one-way exchanges currently use a non-extensible header and encoding defined in section 4 of RFC-5277. These RFCs MUST be updated to enable this draft. These RFCs SHOULD be updated to provide examples

D.1. Implications to RFC-7950

Sections which expose netconf:capability:notification:1.0 are 4.2.10

Sections which provide examples using netconf:notification:1.0 are 7.10.4, 7.16.3, and 9.9.6

D.2. Implications to RFC-8040

Section 6.4 demands use of RFC-5277's netconf:notification:1.0, and later in the section provides an example.

Authors' Addresses

Eric Voit
Cisco Systems

Email: evoit@cisco.com

Henk Birkholz
Fraunhofer SIT

Email: henk.birkholz@sit.fraunhofer.de

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: December 6, 2018

K. Watsen
Juniper Networks
G. Wu
Cisco Networks

June 4, 2018

RESTCONF Client and Server Models
draft-ietf-netconf-restconf-client-server-06

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:

- I-D.ietf-netconf-keystore

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

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

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

- "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:

- "2018-06-04" --> 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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
2. The RESTCONF Client Model 3
  2.1. Tree Diagram 4
  2.2. Example Usage 6
  2.3. YANG Module 8
3. The RESTCONF Server Model 17
  3.1. Tree Diagram 17
  3.2. Example Usage 20
  3.3. YANG Module 23
4. Security Considerations 32
5. IANA Considerations 33

5.1. The IETF XML Registry 33
5.2. The YANG Module Names Registry 34
6. References 34
6.1. Normative References 34
6.2. Informative References 35
Appendix A. Change Log 37
A.1. 00 to 01 37
A.2. 01 to 02 37
A.3. 02 to 03 37
A.4. 03 to 04 37
A.5. 04 to 05 37
A.6. 05 to 06 38
Acknowledgements 38
Authors' Addresses 38

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.

2. The RESTCONF Client Model

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, like that presented in [I-D.ietf-netconf-netconf-client-server], is designed to support any number of possible transports. RESTCONF only supports the TLS transport currently, thus this model only supports the TLS transport.

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

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-restconf-client" module. Just the container is displayed below, but there is also a reuable grouping by the same name that the container is using.

[Note: '\' line wrapping for formatting only]

```
module: ietf-restconf-client
  +--rw restconf-client
     +--rw initiate! {initiate}?
     |  +--rw restconf-server* [name]
     |     +--rw name         string
     |     +--rw endpoints
     |        +--rw endpoint* [name]
     |           +--rw name                 string
     |           +--rw (transport)
     |           |  +--:(tls) {tls-initiate}?
     |           |     +--rw tls
     |           |        +--rw address            inet:host
     |           |        +--rw port?              inet:port-number
     |           |        +--rw client-identity
     |           |        |  +--rw (auth-type)
     |           |        |     +--:(certificate)
     |           |        |        +--rw certificate
     |           |        |           +--rw (local-or-keystore)
     |           |        |              +--:(local)
     |           |        |              |  +--rw algorithm
     |           |        |              |  |       ct:key-algorithm\
-ref
     |           |        |              |  +--rw public-key
     |           |        |              |  |       binary
     |           |        |              |  +--rw private-key
     |           |        |              |  |       union
     |           |        |              |  +--rw cert
     |           |        |              |  |       ct:end-entity-ce\
rt-cms
     |           |        |              |  +---n certificate-expira\
tion
     |           |        |              |     +-- expiration-date?
     |           |        |              |             yang:date-and\
-time
     |           |        |              +--:(keystore)
     |           |        |                       {keystore-implemen\
ted}?
     |           |        |                 +--rw reference
```

```
        |           |        |                        ks:asymmetric-ke\
   y-certificate-ref
        |           |        +--rw server-auth
        |           |        |  +--rw pinned-ca-certs?
        |           |        |  |       ta:pinned-certificates-ref
        |           |        |  +--rw pinned-server-certs?
        |           |        |          ta:pinned-certificates-ref
        |           |        +--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 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 client-identity
                       |  +--rw (auth-type)
                       |     +--:(certificate)
                       |        +--rw certificate
                       |           +--rw (local-or-keystore)
                       |              +--:(local)
                       |              |  +--rw algorithm
                       |              |  |       ct:key-algorithm-ref
                       |              |  +--rw public-key
                       |              |  |       binary
                       |              |  +--rw private-key
```

```
                        |              |  |       union
                        |              |  +--rw cert
                        |              |  |       ct:end-entity-cert-cms\

                        |              |  +---n certificate-expiration
                        |              |     +-- expiration-date?
                        |              |            yang:date-and-time
                        |              +--:(keystore)
                        |                       {keystore-implemented}?
                        |                 +--rw reference
                        |                         ks:asymmetric-key-cert\
   ificate-ref
                        +--rw server-auth
                        |  +--rw pinned-ca-certs?
                        |  |       ta:pinned-certificates-ref
                        |  +--rw pinned-server-certs?
                        |          ta:pinned-certificates-ref
                        +--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].

[Note: '\' line wrapping for formatting only]

```
<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>
        <endpoints>
          <endpoint>
            <name>corp-fw1.example.com</name>
            <tls>
              <address>corp-fw1.example.com</address>
              <client-identity>
```

```
                    <certificate>
                      <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:i\
   etf-crypto-types">ct:secp521r1</algorithm>
                      <private-key>base64encodedvalue==</private-key>
                      <public-key>base64encodedvalue==</public-key>
                      <cert>base64encodedvalue==</cert>
                    </certificate>
                  </client-identity>
                  <server-auth>
                    <pinned-ca-certs>explicitly-trusted-server-ca-certs<\
   /pinned-ca-certs>
                    <pinned-server-certs>explicitly-trusted-server-certs\
   </pinned-server-certs>
                  </server-auth>
                </tls>
              </endpoint>
              <endpoint>
                <name>corp-fw2.example.com</name>
                <tls>
                  <address>corp-fw2.example.com</address>
                  <client-identity>
                    <certificate>
                      <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:i\
   etf-crypto-types">ct:secp521r1</algorithm>
                      <private-key>base64encodedvalue==</private-key>
                      <public-key>base64encodedvalue==</public-key>
                      <cert>base64encodedvalue==</cert>
                    </certificate>
                  </client-identity>
                  <server-auth>
                    <pinned-ca-certs>explicitly-trusted-server-ca-certs<\
   /pinned-ca-certs>
                    <pinned-server-certs>explicitly-trusted-server-certs\
   </pinned-server-certs>
                  </server-auth>
                </tls>
              </endpoint>
            </endpoints>
        </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>
```

```
          <client-identity>
            <certificate>
              <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-cr\
ypto-types">ct:secp521r1</algorithm>
              <private-key>base64encodedvalue==</private-key>
              <public-key>base64encodedvalue==</public-key>
              <cert>base64encodedvalue==</cert>
            </certificate>
          </client-identity>
          <server-auth>
            <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinne\
d-ca-certs>
            <pinned-server-certs>explicitly-trusted-server-certs</pinn\
ed-server-certs>
          </server-auth>
        </tls>
      </endpoint>
    </listen>
</restconf-client>
```

2.3. YANG Module

This YANG module has normative references to [RFC6991], [RFC8040], and [RFC8071], and [I-D.ietf-netconf-tls-client-server].

```
<CODE BEGINS> file "ietf-restconf-client@2018-06-04.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 2018-06-04; // stable grouping definitions
    reference
      "RFC ZZZZ: YANG Groupings for TLS Clients and TLS Servers";
  }

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

```
     contact
      "WG Web:   <http://datatracker.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) 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 "2018-06-04" {
       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 {
       if-feature initiate;
       description
        "The 'tls-initiate' feature indicates that the RESTCONF client
```

```
         supports initiating TLS connections to RESTCONF servers. This
         feature exists as TLS might not be a mandatory to implement
         transport in the future.";
       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 {
       if-feature 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.  This feature exists as
         TLS might not be a mandatory to implement transport in the
         future.";
       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;
         presence "Enables client to initiate TCP connections";
         description
           "Configures client initiating underlying TCP connections.";
         list restconf-server {
           key name;
           min-elements 1;
           description
             "List of RESTCONF servers the RESTCONF client is to
              initiate connections to in parallel.";
```

```
           leaf name {
             type string;
             description
               "An arbitrary name for the RESTCONF server.";
           }
           container endpoints {
             description
               "Container for the list of endpoints.";
             list endpoint {
               key name;
               min-elements 1;
               ordered-by user;
               description
                 "A non-empty user-ordered list of endpoints for this
                  RESTCONF client to try to connect to in sequence.
                  Defining more than one enables high-availability.";
               leaf name {
                 type string;
                 description
                   "An arbitrary name for this endpoint.";
               }
               choice transport {
                 mandatory true;
                 description
                   "Selects between available transports. This is a
                    'choice' statement so as to support additional
                    transport options to be augmented in.";
                 case tls {
                   if-feature tls-initiate;
                   container tls {
                     description
                       "Specifies TLS-specific transport
                        configuration.";
                     leaf address {
                       type inet:host;
                       mandatory true;
                       description
                        "The IP address or hostname of the endpoint.
                         If a domain name is configured, then the
                         DNS resolution should happen on each usage
                         attempt.  If the the DNS resolution results
                         in multiple IP addresses, the IP addresses
                         will be tried according to local preference
                         order until a connection has been established
                         or until all IP addresses have failed.";
                     }
                     leaf port {
                       type inet:port-number;
```

```
                        default 443;
                        description
                         "The IP port for this endpoint. The RESTCONF
                          client will use the IANA-assigned well-known
                          port for 'https' (443) if no value is
                          specified.";
                      }
                      uses ts:tls-client-grouping {
                        refine "client-identity/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 {
                    default persistent-connection;
                    description
                      "Selects between available connection types.";
                    case persistent-connection {
                      container persistent {
                        presence
                         "Indicates that a persistent connection is
                          to be maintained.";
                        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 the underlying TLS session may remain
                             idle.  A TLS 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 TLS
                  server.  An unresponsive 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 TLS server, a TLS-level message will
                   be sent to test the aliveness of the 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 TLS server
                   before assuming the TLS server is no
                   longer alive.";
               }
             }
           }
         }
         case periodic-connection {
           container periodic {
             presence
              "Indicates that a periodic connection is to be
               maintained.";
             description
              "Periodically connect to the RESTCONF server,
```

```
                        so that, e.g., the RESTCONF client can
                        collect data (logs) from the RESTCONF server.
                        Once the connection is closed, for whatever
                        reason, 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 the underlying TLS session may remain
                           idle. A TLS 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.";
                      }
                      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).";
                      }
                    }
                  } // end periodic-connection
                } // end connection-type
              } // end connection-type
              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 reconnect-strategy
              } // end endpoint
            } // end endpoints
          } // end restconf-server
        } // end initiate

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

          leaf idle-timeout {
            type uint16;
```

```
          units "seconds";
          default 3600; // one hour
          description
            "Specifies the maximum number of seconds that an
             underlying TLS session may remain idle. A TLS 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;
          min-elements 1;
          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. This is a
               'choice' statement so as to support additional
               transport options to be augmented in.";
            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 on for incoming call-
                    home connections.  The RESTCONF client will
                    listen on all configured interfaces if no
                    value is specified.  INADDR_ANY (0.0.0.0) or
                    INADDR6_ANY (0:0:0:0:0:0:0:0 a.k.a. ::) MUST
                    be used when the server is to listen on all
                    IPv4 or IPv6 addresses, respectively.";
                }
                leaf port {
                  type inet:port-number;
                  default 4336;
```

```
                    description
                     "The port number to listen on for call-home
                      connections. The RESTCONF client will listen
                      on the IANA-assigned well-known port for
                      'restconf-ch-tls' (4336) if no value is
                       specified.";
                  }
                  uses ts:tls-client-grouping {
                    refine "client-identity/auth-type" {
                      mandatory true;
                      description
                        "RESTCONF clients MUST pass some authentication
                         credentials.";
                    }
                  }
                }
              }
            } // end transport
          } // end endpoint
        } // end listen
      } // end restconf-client
    }
    <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.

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

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-restconf-client" module. Just the container is displayed below, but there is also a reuable grouping by the same name that the container is using.

[Note: '\' line wrapping for formatting only]

```
module: ietf-restconf-server
  +--rw restconf-server
```

```
        +--rw listen! {listen}?
        |  +--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-identity
        |              |  +--rw (local-or-keystore)
        |              |     +--:(local)
        |              |     |  +--rw algorithm
        |              |     |  |       ct:key-algorithm-ref
        |              |     |  +--rw public-key                binary
        |              |     |  +--rw private-key               union
        |              |     |  +--rw cert
        |              |     |  |       ct:end-entity-cert-cms
        |              |     |  +---n certificate-expiration
        |              |     |     +-- expiration-date?
        |              |     |             yang:date-and-time
        |              |     +--:(keystore) {keystore-implemented}?
        |              |        +--rw reference
        |              |                ks:asymmetric-key-certificate-r\
   ef
        |              +--rw client-auth
        |              |  +--rw pinned-ca-certs?
        |              |  |       ta:pinned-certificates-ref
        |              |  +--rw pinned-client-certs?
        |              |  |       ta:pinned-certificates-ref
        |              |  +--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 endpoints
              |  +--rw endpoint* [name]
              |     +--rw name         string
```

```
               |     +--rw (transport)
               |        +--:(tls) {tls-call-home}?
               |           +--rw tls
               |              +--rw address            inet:host
               |              +--rw port?              inet:port-number
               |              +--rw server-identity
               |              |  +--rw (local-or-keystore)
               |              |     +--:(local)
               |              |     |  +--rw algorithm
               |              |     |  |       ct:key-algorithm-ref
               |              |     |  +--rw public-key
               |              |     |  |       binary
               |              |     |  +--rw private-key
               |              |     |  |       union
               |              |     |  +--rw cert
               |              |     |  |       ct:end-entity-cert-cms
               |              |     |  +---n certificate-expiration
               |              |     |     +-- expiration-date?
               |              |     |             yang:date-and-time
               |              |     +--:(keystore) {keystore-implemented\
   }?
               |              |        +--rw reference
               |              |                ks:asymmetric-key-certifi\
   cate-ref
               |              +--rw client-auth
               |              |  +--rw pinned-ca-certs?
               |              |  |       ta:pinned-certificates-ref
               |              |  +--rw pinned-client-certs?
               |              |  |       ta:pinned-certificates-ref
               |              |  +--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 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].

[Note: '\' line wrapping for formatting only]

```
<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"\
>

  <!-- endpoints to listen for RESTCONF connections on -->
  <listen>
    <endpoint>
      <name>netconf/tls</name>
      <tls>
        <address>11.22.33.44</address>
        <server-identity>
          <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-cryp\
to-types">ct:secp521r1</algorithm>
          <private-key>base64encodedvalue==</private-key>
          <public-key>base64encodedvalue==</public-key>
          <cert>base64encodedvalue==</cert>
        </server-identity>
        <client-auth>
          <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinne\
d-ca-certs>
          <pinned-client-certs>explicitly-trusted-client-certs</pinn\
ed-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>

    <!-- call home to a RESTCONF client with two endpoints -->
    <call-home>
      <restconf-client>
        <name>config-manager</name>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <tls>
              <address>22.33.44.55</address>
              <server-identity>
                <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-\
   crypto-types">ct:secp521r1</algorithm>
                <private-key>base64encodedvalue==</private-key>
                <public-key>base64encodedvalue==</public-key>
                <cert>base64encodedvalue==</cert>
              </server-identity>
              <client-auth>
                <pinned-ca-certs>explicitly-trusted-client-ca-certs</p\
   inned-ca-certs>
                <pinned-client-certs>explicitly-trusted-client-certs</\
   pinned-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>
          <endpoint>
            <name>west-data-center</name>
            <tls>
              <address>33.44.55.66</address>
              <server-identity>
                <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-\
   crypto-types">ct:secp521r1</algorithm>
                <private-key>base64encodedvalue==</private-key>
                <public-key>base64encodedvalue==</public-key>
                <cert>base64encodedvalue==</cert>
              </server-identity>
              <client-auth>
                <pinned-ca-certs>explicitly-trusted-client-ca-certs</p\
   inned-ca-certs>
                <pinned-client-certs>explicitly-trusted-client-certs</\
   pinned-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>
        </endpoints>
        <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 Module

This YANG module has normative references to [RFC6991], [RFC7407], [RFC8040], [RFC8071], and [I-D.ietf-netconf-tls-client-server].

```
<CODE BEGINS> file "ietf-restconf-server@2018-06-04.yang"
module ietf-restconf-server {
  yang-version 1.1;

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

  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 2018-06-04; // stable grouping definitions
    reference
      "RFC ZZZZ: YANG Groupings for TLS Clients and TLS Servers";
  }

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

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

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

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

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


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

       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 "2018-06-04" {
       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 {
       if-feature listen;
       description
        "The 'tls-listen' feature indicates that the RESTCONF server
         supports opening a port to listen for incoming RESTCONF
         client connections.  This feature exists as TLS might not
         be a mandatory to implement transport in the future.";
       reference
        "RFC 8040: RESTCONF Protocol";
     }
```

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

     feature tls-call-home {
       if-feature call-home;
       description
        "The 'tls-call-home' feature indicates that the RESTCONF
         server supports initiating connections to RESTCONF clients.
         This feature exists as TLS might not be a mandatory to
         implement transport in the future.";
       reference
        "RFC 8071: NETCONF Call Home and RESTCONF Call Home";
     }

     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;
         presence "Enables server to listen for TCP connections";
         description "Configures listen behavior";
         list endpoint {
           key name;
           min-elements 1;
           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. This is a
                 'choice' statement so as to support additional
                 transport options to be augmented in.";
              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 on for incoming
                     connections.  The RESTCONF server will listen
                     on all configured interfaces if no value is
                     specified.  INADDR_ANY (0.0.0.0) or INADDR6_ANY
                     (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be used when
                     the server is to listen on all IPv4 or IPv6
                     addresses, respectively.";
                  }
                  leaf port {
                    type inet:port-number;
                    default 443;
                    description
                     "The local port number to listen on.  If no value
                      is specified, the IANA-assigned port value for
                      'https' (443) is used.";
                  }
                  uses ts:tls-server-grouping {
                    refine "client-auth" {
                      must 'pinned-ca-certs or pinned-client-certs';
                      description
                        "RESTCONF servers MUST be able to validate
                         clients.";
                    }
                    augment "client-auth" {
                      description
                        "Augments in the cert-to-name structure,
                         so the RESTCONF server can map TLS-layer
                         client certificates to RESTCONF usernames.";
                      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 7407: A YANG Data Model for SNMP
                                   Configuration.";
                    }
                  }
                }
              } // end tls container
            } // end tls case
          } // end transport
        } // end endpoint
      } // end listen

      container call-home {
        if-feature call-home;
        presence "Enables server to initiate TCP connections";
        description "Configures call-home behavior";
        list restconf-client {
          key name;
          min-elements 1;
          description
            "List of RESTCONF clients the RESTCONF server is to
             initiate call-home connections to in parallel.";
          leaf name {
            type string;
            description
              "An arbitrary name for the remote RESTCONF client.";
          }
          container endpoints {
            description
              "Container for the list of endpoints.";
            list endpoint {
              key name;
              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.";
              }
              choice transport {
                mandatory true;
```

```
                  description
                    "Selects between available transports. This is a
                     'choice' statement so as to support additional
                     transport options to be augmented in.";
                  case tls {
                    if-feature tls-call-home;
                    container tls {
                      description
                        "Specifies TLS-specific call-home transport
                         configuration.";
                      leaf address {
                        type inet:host;
                        mandatory true;
                        description
                         "The IP address or hostname of the endpoint.
                          If a domain name is configured, then the
                          DNS resolution should happen on each usage
                          attempt.  If the DNS resolution results in
                          multiple IP addresses, the IP addresses will
                          be tried according to local preference order
                          until a connection has been established or
                          until all IP addresses have failed.";
                      }
                      leaf port {
                        type inet:port-number;
                        default 4336;
                        description
                         "The IP port for this endpoint.  The RESTCONF
                          server will use the IANA-assigned well-known
                          port for 'restconf-ch-tls' (4336) if no value
                          is specified.";
                      }
                      uses ts:tls-server-grouping {
                        refine "client-auth" {
                         must 'pinned-ca-certs or pinned-client-certs';
                         description
                          "RESTCONF servers MUST be able to validate
                           clients.";
                        }
                        augment "client-auth" {
                          description
                           "Augments in the cert-to-name structure,
                            so the RESTCONF server can map TLS-layer
                            client certificates to RESTCONF usernames.";
                          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 7407: A YANG Data Model for SNMP
                               Configuration.";
                          }
                        }
                      }
                    }
                  }
                } // end transport
              } // end endpoint
            } // end endpoints
            container connection-type {
              description
               "Indicates the RESTCONF client's preference for how the
                RESTCONF server's connection is maintained.";
              choice connection-type {
                default persistent-connection;
                description
                  "Selects between available connection types.";
                case persistent-connection {
                  container persistent {
                    presence
                     "Indicates that a persistent connection is to be
                      maintained.";
                    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.";
                    leaf idle-timeout {
                      type uint32;
                      units "seconds";
                      default 86400;  // one day;
                      description
                        "Specifies the maximum number of seconds that
```

```
                        the underlying TLS session may remain idle.
                        A TLS 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 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
                     "Indicates that a periodic connection is to be
                      maintained.";
```

```
                    description
                     "Periodically connect to the RESTCONF client, so
                      that the RESTCONF client may send requests pending
                      for the RESTCONF server.  Once the connection has
                      been closed, for whatever reason, the 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
                         the underlying TLS session may remain idle.
                         A TLS 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
                       "The maximum amount of unconnected time
                        the RESTCONF server will wait before
                        re-establishing a connection to the
                        RESTCONF client.  The RESTCONF server
                        may initiate a connection to the RESTCONF
                        client before this time if desired
                        (e.g., to deliver a notification).";
                    }
                  }
                }
              }
            }
            container reconnect-strategy {
              description
               "The reconnection strategy directs how a RESTCONF
                server reconnects to a RESTCONF client after after
                discovering its connection to the client has dropped,
                even if due to a reboot.  The RESTCONF 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 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
                       servers SHOULD be able to remember the last
                       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).";
              }
            }
          }
        }
      }
    }
    <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:

/: The entire data trees defined by the modules defined in this draft are 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

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. References

6.1. Normative References

[I-D.ietf-netconf-keystore]
Watsen, K., "YANG Data Model for a "Keystore" Mechanism", draft-ietf-netconf-keystore-04 (work in progress), October 2017.

[I-D.ietf-netconf-tls-client-server]
Watsen, K. and G. Wu, "YANG Groupings for TLS Clients and TLS Servers", draft-ietf-netconf-tls-client-server-05 (work in progress), October 2017.

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

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

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

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

6.2. Informative References

[I-D.ietf-netconf-netconf-client-server]
Watsen, K. and G. Wu, "NETCONF Client and Server Models", draft-ietf-netconf-netconf-client-server-05 (work in progress), October 2017.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6536>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Change Log

A.1. 00 to 01

- Renamed "keychain" to "keystore".

A.2. 01 to 02

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

A.3. 02 to 03

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

A.4. 03 to 04

- Added RFC 8174 to Requirements Language Section.
- Replaced refine statement in ietf-restconf-client to add a mandatory true.
- Added refine statement in ietf-restconf-server to add a must statement.
- Now there are containers and groupings, for both the client and server models.
- Now tree diagrams reference ietf-netmod-yang-tree-diagrams
- Updated examples to inline key and certificates (no longer a leafref to keystore)

A.5. 04 to 05

- Now tree diagrams reference ietf-netmod-yang-tree-diagrams
- Updated examples to inline key and certificates (no longer a leafref to keystore)

A.6. 05 to 06

- o Fixed change log missing section issue.
- o Updated examples to match latest updates to the crypto-types, trust-anchors, and keystore drafts.
- o Reduced line length of the YANG modules to fit within 69 columns.

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.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Networks

EMail: garywu@cisco.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: December 20, 2018

E. Voit
R. Rahman
E. Nilsen-Nygaard
Cisco Systems
A. Clemm
Huawei
A. Bierman
YumaWorks
June 18, 2018

RESTCONF and HTTP Transport for Event Notifications
draft-ietf-netconf-restconf-notif-06

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 publisher defined event streams or nodes/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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 20, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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. Dynamic Subscription 3
3.1. Transport Connectivity 4
3.2. Discovery 4
3.3. RESTCONF RPCs and HTTP Status Codes 4
3.4. Call Flow for HTTP2 6
3.5. Call flow for HTTP1.1 8
4. Configured Subscription 10
4.1. Transport Connectivity 10
4.2. Call Flow 11
5. QoS Treatment 12
6. Mandatory JSON and datastore support 12
7. Notification Messages 12
8. YANG Tree 12
9. YANG module 13
10. IANA Considerations 15
11. Security Considerations 15
12. Acknowledgments 16
13. References 16
13.1. Normative References 17
13.2. Informative References 18
Appendix A. RESTCONF over GRPC 19
Appendix B. Examples 19
B.1. Dynamic Subscriptions 19
B.1.1. Establishing Dynamic Subscriptions 19
B.1.2. Modifying Dynamic Subscriptions 22
B.1.3. Deleting Dynamic Subscriptions 23
B.2. Configured Subscriptions 24
B.2.1. Creating Configured Subscriptions 25
B.2.2. Modifying Configured Subscriptions 27
B.2.3. Deleting Configured Subscriptions 29
B.3. Subscription State Notifications 30
B.3.1. subscription-started and subscription-modified . . . 30
B.3.2. subscription-completed, subscription-resumed, and replay-complete 31
B.3.3. subscription-terminated and subscription-suspended . 31
Appendix C. Changes between revisions 32
Authors' Addresses 33

1. Introduction

Mechanisms to support event subscription and push are defined in [I-D.draft-ietf-netconf-subscribed-notifications]. Enhancements to [I-D.draft-ietf-netconf-subscribed-notifications] which enable YANG datastore subscription and push are defined in [I-D.ietf-netconf-yang-push]. This document provides a transport specification for these protocols over RESTCONF [RFC8040] and HTTP. Driving these requirements is [RFC7923].

The streaming of notifications encapsulating the resulting information push can be done with either HTTP1.1 [RFC7231] or HTTP2 [RFC7540].

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 [I-D.draft-ietf-netconf-subscribed-notifications]: configured subscription, dynamic subscription, event stream, notification message, publisher, receiver, subscriber, and subscription.

Other terms reused include datastore, which is defined in [RFC8342], and HTTP2 stream which maps to the definition of "stream" within [RFC7540], Section 2.

[note to the RFC Editor - please replace XXXX within this document with the number of this document]

3. Dynamic Subscription

This section provides specifics on how to establish and maintain dynamic subscriptions over HTTP 1.1 and HTTP2 via signaling messages transported over RESTCONF [RFC8040]. Subscribing to event streams is accomplished in this way via a RESTCONF POST into RPCs defined within [I-D.draft-ietf-netconf-subscribed-notifications] Section 2.4. YANG datastore subscription is accomplished via augmentations to [I-D.draft-ietf-netconf-subscribed-notifications] as described within [I-D.ietf-netconf-yang-push] Section 4.4.

Common across all HTTP based dynamic subscriptions is that a POST needs to be made against a specific URI on the Publisher. Subscribers cannot pre-determine the URI against which a subscription might exist on a publisher, as the URI will only exist after the "establish-subscription" has been accepted. The subscription URI

will be determined and sent as part of the response to the "establish-subscription", and a subsequent POST to this URI will be done in order to start the flow of notification messages back to the subscriber. A subscription does not move to the active state as per Section 2.4.1. of [I-D.draft-ietf-netconf-subscribed-notifications] until the POST is received.

3.1. Transport Connectivity

For a dynamic subscription, where an HTTP client session doesn’t already exist, a new client session is initiated from the subscriber. If the subscriber is unsure if HTTP2 is supported by the publisher, HTTP1.1 will be used for initial messages, and these messages will include an HTTP version upgrade request as per [RFC7230], Section 6.7. If a publisher response indicates that HTTP2 is supported, HTTP2 will be used between subscriber and publisher for future HTTP interactions as per [RFC7540].

A subscriber SHOULD establish the HTTP session over TLS [RFC5246] in order to secure the content in transit.

Without the involvement of additional protocols, neither HTTP1.1 nor HTTP2 sessions by themselves allow for a quick recognition of when the communication path has been lost with the publisher. Where quick recognition of the loss of a publisher is required, a subscriber SHOULD connect over TLS [RFC5246], and use a TLS heartbeat [RFC6520] to track HTTP session continuity. In the case where a TLS heartbeat is included, it should be sent just from receiver to publisher. Loss of the heartbeat MUST result in any subscription related TCP sessions between those endpoints being torn down. A subscriber can then attempt to re-establish.

3.2. Discovery

Subscribers can learn what event streams a RESTCONF server supports by querying the "streams" container of ietf-subscribed-notification.yang. Subscribers can learn what datastores a RESTCONF server supports by following [I-D.draft-ietf-netconf-nmda-restconf].

3.3. RESTCONF RPCs and HTTP Status Codes

Specific HTTP responses codes as defined in [RFC7231] section 6 will indicate the result of RESTCONF RPC requests with publisher. An HTTP status code of 200 is the proper response to any successful RPC defined within [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push].

If a publisher fails to serve the RPC request for one of the reasons indicated in [I-D.draft-ietf-netconf-subscribed-notifications] Section 2.4.6 or [I-D.ietf-netconf-yang-push] Appendix A, this will be indicated by "406" status code transported in the HTTP response.

When a "406" status code is returned, the RPC reply MUST include an "rpc-error" element per [RFC8040] Section 7.1 with the following parameter values:

- o an "error-type" node of "application".

- o an "error-tag" node of "operation-failed".

- o an "error-app-tag" node with the value being a string that corresponds to an identity associated with the error, as defined in [I-D.draft-ietf-netconf-subscribed-notifications] section 2.4.6 for general subscriptions, and [I-D.ietf-netconf-yang-push] Appendix A.1, for datastore subscriptions. The tag to use depends on the RPC for which the error occurred. Viable errors for different RPCs are as follows:

RPC	select an identity with a base
establish-subscription	establish-subscription-error
modify-subscription	modify-subscription-error
delete-subscription	delete-subscription-error
kill-subscription	kill-subscription-error
resynch-subscription	resynch-subscription-error

Each error identity will be inserted as the "error-app-tag" using JSON encoding following the form <modulename>:<identityname>. An example of such as valid encoding would be "ietf-subscribed-notifications:no-such-subscription".

- o In case of error responses to an "establish-subscription" or "modify-subscription" request there is the option of including an "error-info" node. This node may contain hints for parameter settings that might lead to successful RPC requests in the future. Following are the yang-data structures which may be returned:

```
establish-subscription returns hints in yang-data structure
---------------------- ------------------------------------
target: event stream   establish-subscription-stream-error-info
target: datastore      establish-subscription-datastore-error-info

modify-subscription    returns hints in yang-data structure
---------------------- ------------------------------------
target: event stream   modify-subscription-stream-error-info
target: datastore      modify-subscription-datastore-error-info
```

The yang-data included within "error-info" SHOULD NOT include the optional leaf "error-reason", as such a leaf would be redundant with information that is already placed within the "error-app-tag".

In case of an rpc error as a result of a "delete-subscription", a "kill-subscription", or a "resynch-subscription" request, no "error-info" needs to be included, as the "subscription-id" is the only RPC input parameter and no hints regarding this RPC input parameters need to be provided.

Note that "error-path" does not need to be included with the "rpc-error" element, as subscription errors are generally not associated with nodes in the datastore but with the choice of RPC input parameters.

3.4. Call Flow for HTTP2

Requests to [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push] augmented RPCs are sent on one or more HTTP2 streams indicated by (a) in Figure 1. A successful "establish-subscription" will result in an RPC response returned with both a subscription identifier which uniquely identifies a subscription, as well as a URI which uniquely identifies the location of subscription on the publisher. This URI is defined via the "uri" leaf the Data Model in Section 9.

An HTTP POST is then sent on a logically separate HTTP2 stream (b) to the URI on the publisher. This initiates to initiate the flow of notification messages which are sent in HTTP Data frames as a response to the POST. In the case below, a newly established subscription has its associated notification messages pushed over HTTP2 stream (7). These notification messages are placed into a HTTP2 Data frame (see [RFC7540] Section 6.1).

```
   +------------+                                +------------+
   | Subscriber |                                | Publisher  |
   |HTTP2 Stream|                                |HTTP2 Stream|
   |  (a)  (b)  |                                |  (a)  (b)  |
   +------------+                                +------------+
       | RESTCONF POST (RPC:establish-subscription)   |
       |---------------------------------------------->|
       |                          HTTP 200 OK (ID,URI)|
       |<----------------------------------------------|
       |   (7)HTTP POST (URI)                            (7)
       |    |---------------------------------------------->|
       |    |                                   HTTP 200 OK|
       |    |<----------------------------------------------|
       |    |                     HTTP Data (notif-message)|
       |    |<----------------------------------------------|
       | RESTCONF POST (RPC:modify-subscription)      |    |
       |---------------------------------------------->|    |
       |    |                               HTTP 200 OK|    |
       |<----------------------------------------------|    |
       |    |             HTTP Data (subscription-modified)|
       |    |<----------------------------------------(c)|
       |    |                     HTTP Data (notif-message)|
       |    |<----------------------------------------------|
       | RESTCONF POST (RPC:delete-subscription)      |    |
       |---------------------------------------------->|    |
       |    |                               HTTP 200 OK|    |
       |<----------------------------------------------|    |
       |    |                  HTTP Headers (end of stream)|
       |   (/7)<----------------------------------------(/7)
       |
```

Figure 1: Dynamic with HTTP2

Additional requirements for dynamic subscriptions over HTTP2 include:

- o A unique HTTP2 stream MAY be used for each subscription.

- o A single HTTP2 stream MUST NOT be used for subscriptions with different DSCP values.

- o All subscription state notifications from a publisher MUST be returned in a separate HTTP Data frame within the HTTP2 stream used by the subscription to which the state change refers.

- o In addition to an RPC response for a "modify-subscription" RPC traveling over (a), a "subscription-modified" state change notification must be sent within HTTP2 stream (b). This allows the receiver to know exactly when the new terms of the

subscription have been applied to the notification messages. See arrow (c).

- o Additional RPCs for a particular subscription MUST NOT use the HTTP2 stream currently providing notification messages subscriptions.

- o An HTTP end of stream message MUST not be sent until all subscriptions using that HTTP2 stream have completed.

3.5. Call flow for HTTP1.1

The call flow is defined in Figure 2. Requests to [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push] augmented RPCs are sent on a TCP connection indicated by (a). A successful "establish-subscription" will result in an RPC response returned with both a subscription identifier which uniquely identifies a subscription, as well as a URI which uniquely identifies the location of subscription on the publisher (b). This URI is defined via the "uri" leaf the Data Model in Section 9.

An HTTP POST is then sent on a logically separate TCP connection (b) to the URI on the publisher. This initiates to initiate the flow of notification messages which are sent in SSE [W3C-20150203] as a response to the POST.

```
   +--------------+                            +--------------+
   |  Subscriber  |                            |   Publisher  |
   |TCP connection|                            |TCP connection|
   |  (a)  (b)    |                            |    (a)  (b)  |
   +--------------+                            +--------------+
       | RESTCONF POST (RPC:establish-subscription)   |
       |---------------------------------------------->|
       |                          HTTP 200 OK (ID,URI)|
       |<----------------------------------------------|
       |    |HTTP GET (URI)                                |
       |    |---------------------------------------------->|
       |    |                                   HTTP 200 OK|
       |    |<----------------------------------------------|
       |    |                            SSE (notif-message)|
       |    |<----------------------------------------------|
       | RESTCONF POST (RPC:modify-subscription)      |    |
       |---------------------------------------------->|    |
       |    |                                HTTP 200 OK|    |
       |<----------------------------------------------|    |
       |    |                   SSE (subscription-modified)|
       |    |<----------------------------------------(c)|
       |    |                            SSE (notif-message)|
       |    |<----------------------------------------------|
       | RESTCONF POST (RPC:delete-subscription)      |    |
       |---------------------------------------------->|    |
       |    |                                HTTP 200 OK|    |
       |<----------------------------------------------|    |
       |    |                                              |
       |    |
```

Figure 2: Dynamic with HTTP1.1

Additional requirements for dynamic subscriptions over HTTP1.1 include:

- o All subscription state notifications from a publisher MUST be returned in a separate SSE message used by the subscription to which the state change refers.

- o Subscription RPCs MUST NOT use the TCP connection currently providing notification messages for that subscription.

- o In addition to an RPC response for a "modify-subscription" RPC traveling over (a), a "subscription-modified" state change notification must be sent within stream (b). This allows the receiver to know exactly when the new terms of the subscription have been applied to the notification messages. See arrow (c).

Open question, should we just eliminate this possibility of HTTP1.1 for subscriptions? It would make the design simpler.

4. Configured Subscription

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 notification messages to the receiver. Since RESTCONF might not exist on the receiver, it is not desirable to require that subscribed content be pushed with any dependency on RESTCONF. Therefore in place of RESTCONF, an HTTP2 Client connection must be established with an HTTP2 Server located on the receiver. Notification messages will then be sent as part of an extended HTTP POST to the receiver.

4.1. Transport Connectivity

Configured subscriptions MUST only be connected over HTTP2 via a client session initiated from the publisher. Following are the conditions which MUST be met before establishing a new HTTP2 connection with a receiver:

- o a configured subscription has a receiver in the connecting state as described in [I-D.draft-ietf-netconf-subscribed-notifications], section 2.5.1.,

- o the transport configured for that subscription is HTTP2,

- o there are state change notifications or notification messages pending for that receiver, and

- o no HTTP2 transport session exists to that receiver,

If the above conditions are met, then the publisher MUST initiate a transport session via RESTCONF call home [RFC8071], section 4.1 to that receiver. HTTP2 only communications must be used as per [RFC7540], Section 3.3 when the HTTP session over TLS [RFC5246]. and [RFC7540], Section 3.4 when transporting cleartext over TCP. Note that a subscriber SHOULD establish over TLS in order to secure the content in transit.

If the RESTCONF call home fails because the publisher receives receiver credentials which are subsequently declined per [RFC8071], Section 4.1, step S5 authentication, then that receiver MUST be placed into the timeout state.

If the call home fails to establish for any other reason, the publisher MUST NOT progress the receiver to the active state. Additionally, the publisher SHOULD place the receiver into the timeout state after a predetermined number of either failed call home attempts or remote transport session termination by the receiver.

4.2. Call Flow

With HTTP2 connectivity established, a POST of each new "subscription-started" state change notification messages will be addressed to HTTP augmentation code on the receiver capable of accepting and acknowledging to subscription state change notifications. Until the "HTTP 200 OK" at point (c) of Figure 3 for each the "subscription-started" state change notification, a publisher MUST NOT progress the receiver to the active state. In other words, is at point (c) which indicates that the receiver is ready for the delivery of subscribed content. At this point a notification-messages including subscribed content may be placed onto an HTTP2 stream for that subscription.

```
+------------+                                +------------+
|  Receiver  |                                | Publisher  |
|HTTP2 Stream|                                |HTTP2 Stream|
|  (a)  (b)  |                                |  (a)  (b)  |
+------------+                                +------------+
    |HTTP Post Headers, Data (subscription-started)|
    |<---------------------------------------------|
    | HTTP 200 OK                                  |
    |--------------------------------------------->(c)
    |    |       HTTP Post Headers, Data (notif-message)|
    |    |<---------------------------------------------|
    |    |                     HTTP Data (notif-message)|
    |    |<---------------------------------------------|
    |    |                    HTTP Data (sub-terminated)|
    |    |<---------------------------------------------|
    |    |HTTP 200 OK                                   |
    |    |--------------------------------------------->|
```

Figure 3: Configured over HTTP2

Additional requirements for configured subscriptions over HTTP2 include:

- o A unique HTTP2 stream MAY be used for each subscription.

- o A single HTTP2 stream MUST NOT be used for subscriptions with different DSCP values.

- o All subscription state notifications from a publisher MUST be returned in a separate HTTP Data frame within the HTTP2 stream used by the subscription to which the state change refers.

- o An HTTP end of stream message MUST not be sent until all subscriptions using that HTTP2 stream have completed.

5. QoS Treatment

To meet subscription quality of service promises, the publisher MUST take any existing subscription "dscp" and apply it to the DSCP marking in the IP header.

In addition, where HTTP2 transport is available to a notification message queued for transport to a receiver, the publisher MUST:

- o take any existing subscription "priority" and copy it into the HTTP2 stream priority, and

- o take any existing subscription "dependency" and map the HTTP2 stream for the parent subscription into the HTTP2 stream dependency.

6. Mandatory JSON and datastore support

A publisher supporting [I-D.ietf-netconf-yang-push] MUST support the "operational" datastore as defined by [RFC8342].

The "encode-json" feature of [I-D.draft-ietf-netconf-subscribed-notifications] is mandatory to support. This indicates that JSON is a valid encoding for RPCs, state change notifications, and subscribed content.

7. Notification Messages

Notification messages transported over HTTP will be encoded using one-way operation schema defined within [RFC5277], section 4.

8. YANG Tree

The YANG model defined in Section 9 has one leaf augmented into four places of [I-D.draft-ietf-netconf-subscribed-notifications], plus two identities. As the resulting full tree is large, it will only be inserted at later stages of this document.

9. YANG module

This module references
[I-D.draft-ietf-netconf-subscribed-notifications].

```
<CODE BEGINS> file "ietf-http-subscribed-notifications@2018-06-11.yang"
module ietf-http-subscribed-notifications {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-http-subscribed-notifications";

  prefix hsn;

  import ietf-subscribed-notifications {
    prefix sn;
  }
  import ietf-inet-types {
    prefix inet;
  }

  organization "IETF NETCONF (Network Configuration) Working Group";
  contact
    "WG Web:   <http:/tools.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>

     Editor:   Eric Voit
               <mailto:evoit@cisco.com>

     Editor:   Alexander Clemm
               <mailto:ludwig@clemm.org>

     Editor:   Reshad Rahman
               <mailto:rrahman@cisco.com>";

  description
    "Defines HTTP variants as a supported transports for subscribed
    event notifications.

    Copyright (c) 2018 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
    (https://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 2018-06-11 {
     description
       "Initial version";
     reference
       "RFC XXXX: RESTCONF and HTTP Transport for Event Notifications";
   }

   identity http2 {
     base sn:transport;
     base sn:inline-address;
         base sn:configurable-encoding;
     description
       "HTTP2 is used a transport for notification messages and state
        change notifications.";
   }

   identity http1.1 {
     base sn:transport;
     base sn:inline-address;
         base sn:configurable-encoding;
     description
       "HTTP1.1 is used a transport for notification messages and state
        change notifications.";
   }

   grouping uri {
     description
       "Provides a reusable description of a URI.";
     leaf uri {
       type inet:uri;
       config false;
       description
         "Location of a subscription specific URI on the publisher.";
     }
   }

   augment "/sn:establish-subscription/sn:output" {
     description
       "This augmentation allows HTTP specific parameters for a
       response to a publisher’s subscription request.";
     uses uri;
   }

   augment "/sn:subscriptions/sn:subscription" {
     description
       "This augmentation allows HTTP specific parameters to be
```

```
        exposed for a subscription.";
      uses uri;
    }

    augment "/sn:subscription-started" {
      description
        "This augmentation allows HTTP specific parameters to be included
        part of the notification that a subscription has started.";
      uses uri;
    }

    augment "/sn:subscription-modified" {
      description
        "This augmentation allows HTTP specific parameters to be included
        part of the notification that a subscription has been modified.";
      uses uri;
    }

  }
  <CODE ENDS>
```

10. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-http-subscribed-notifications
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-http-subscribed-notifications
Namespace: urn:ietf:params:xml:ns:yang:ietf-http-subscribed-notifications
Prefix: hsn
Reference: RFC XXXX: RESTCONF and HTTP Transport for Event Notifications

11. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management transports 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 layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The one new data node introduced 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 this data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

Container: "/subscriptions"

o "uri": leaf will show where subscribed resources might be located on a publisher. Access control must be set so that only someone with proper access permissions, and perhaps even HTTP session has the ability to access this resource.

One or more publishers of configured subscriptions could be used to overwhelm a receiver which doesn’t even support subscriptions. There are two protections needing support on a publisher. First, notification messages for configured subscriptions MUST only be transmittable over encrypted transports. Clients which do not want pushed content 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 subscribed content.

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.

The NETCONF Authorization Control Model [RFC6536] SHOULD be used to control and restrict authorization of subscription configuration.

12. Acknowledgments

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

13. References

13.1. Normative References

[GRPC] "RPC framework that runs over HTTP2", August 2017, <https://grpc.io/>.

[I-D.draft-ietf-netconf-subscribed-notifications] Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-13 (work in progress), April 2018.

[I-D.ietf-netconf-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/>.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6242>.

[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, <https://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, <https://www.rfc-editor.org/info/rfc6536>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[W3C-20150203] "Server-Sent Events, World Wide Web Consortium CR CR-eventsource-20121211", February 2015, <https://www.w3.org/TR/2015/REC-eventsource-20150203/>.

13.2. Informative References

[I-D.draft-ietf-netconf-netconf-event-notifications] Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Nilsen-Nygaard, E., and A. Tripathy, "NETCONF support for event notifications", May 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-event-notifications/>.

[I-D.draft-ietf-netconf-nmda-restconf]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "RESTCONF Extensions to Support the Network Management Datastore Architecture", April 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-nmda-restconf/>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <https://www.rfc-editor.org/info/rfc7923>.

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

Appendix A. RESTCONF over GRPC

An initial goal for this document was to support [GRPC] transport seamlessly without any mapping or extra layering. However there is an incompatibility of RESTCONF and GRPC. RESTCONF uses HTTP GET, and GRPC uses HTTP2’s POST rather than GET. As GET is used across RESTCONF for things like capabilities exchange, a seamless mapping depends on specification changes outside the scope of this document. If/when GRPC supports GET, or RESTCONF is updated to support POST, this should be revisited. It is hoped that the resulting fix will be transparent to this document.

Appendix B. Examples

This section is non-normative. To allow easy comparison, this section mirrors the functional examples shown with NETCONF over XML within [I-D.draft-ietf-netconf-netconf-event-notifications]. In addition, HTTP2 vs HTTP1.1 headers are not shown as the contents of the JSON encoded objects are identical within.

B.1. Dynamic Subscriptions

B.1.1. Establishing Dynamic Subscriptions

The following figure shows two successful "establish-subscription" RPC requests as per [I-D.draft-ietf-netconf-subscribed-notifications]. The first request

is given a subscription identifier of 22, the second, an identifier of 23.

```
   +------------+                  +-----------+
   | Subscriber |                  | Publisher |
   +------------+                  +-----------+
         |                               |
         |establish-subscription         |
         |------------------------------>|  (a)
         |     HTTP 200 OK, id#22, URI#1 |
         |<------------------------------|  (b)
         |POST (URI#1)                   |
         |------------------------------>|  (c)
         | HTTP 200 OK,notif-mesg (id#22)|
         |<------------------------------|
         |                               |
         |                               |
         |establish-subscription         |
         |------------------------------>|
         |      HTTP 200 OK, id#23, URI#2|
         |<------------------------------|
         |POST (URI#2)                   |
         |------------------------------>|
         |                               |
         |                               |
         |             notif-mesg (id#22)|
         |<------------------------------|
         | HTTP 200 OK,notif-mesg (id#23)|
         |<------------------------------|
         |                               |
```

Figure 4: Multiple subscriptions over RESTCONF/HTTP

To provide examples of the information being transported, example messages for interactions in Figure 4 are detailed below:

```
POST /restconf/operations/subscriptions:establish-subscription

{
   "ietf-subscribed-notifications:input": {
      "stream": "NETCONF",
      "stream-xpath-filter": "/ex:foo/",
      "dscp": "10"
   }
}
```

Figure 5: establish-subscription request (a)

As publisher was able to fully satisfy the request, the publisher sends the subscription identifier of the accepted subscription, and the URI:

```
HTTP status code - 200

{
   "identifier": "22",
   "uri": "/subscriptions/22"
}
```

Figure 6: establish-subscription success (b)

Upon receipt of the successful response, the subscriber POSTs to the provided URI to start the flow of notification messages. When the publisher receives this, the subscription is moved to the active state (c).

```
POST /restconf/operations/subscriptions/22
```

Figure 7: establish-subscription subsequent POST

While not shown in Figure 4, if the publisher had not been able to fully satisfy the request, or subscriber has no authorization to establish the subscription, the publisher would have sent an RPC error response. For instance, if the "dscp" value of 10 asserted by the subscriber in Figure 5 proved unacceptable, the publisher may have returned:

```
HTTP status code - 406

{ "ietf-restconf:errors" : {
    "error" : [
      {
        "error-type": "application",
        "error-tag": "operation-failed",
        "error-severity": "error",
        "error-app-tag":
            "ietf-subscribed-notifications:dscp-unavailable"
      }
    ]
  }
}
```

Figure 8: an unsuccessful establish subscription

The subscriber can use this information in future attempts to establish a subscription.

B.1.2. Modifying Dynamic Subscriptions

An existing subscription may be modified. The following exchange shows a negotiation of such a modification via several exchanges between a subscriber and a publisher. This negotiation consists of a failed RPC modification request/response, followed by a successful one.

```
+------------+                 +-----------+
| Subscriber |                 | Publisher |
+------------+                 +-----------+
      |                            |
      |  notification message (id#23)|
      |<----------------------------|
      |                            |
      |modify-subscription (id#23)   |
      |---------------------------->|  (d)
      |    HTTP 406 error (with hint)|
      |<----------------------------|  (e)
      |                            |
      |modify-subscription (id#23)   |
      |---------------------------->|
      |                  HTTP 200 OK |
      |<----------------------------|
      |                            |
      |            notif-mesg (id#23)|
      |<----------------------------|
      |                            |
```

Figure 9: Interaction model for successful subscription modification

If the subscription being modified in Figure 9 is a datastore subscription as per [I-D.ietf-netconf-yang-push], the modification request made in (d) may look like that shown in Figure 10. As can be seen, the modifications being attempted are the application of a new xpath filter as well as the setting of a new periodic time interval.

```
POST /restconf/operations/subscriptions:modify-subscription

{
 "ietf-subscribed-notifications:input": {
    "identifier": "23",
    "ietf-yang-push:datastore-xpath-filter":
      "/interfaces-state/interface/oper-status"
      "ietf-yang-push:periodic": {
       "ietf-yang-push:period": "500"
      }
 }
}
```

Figure 10: Subscription modification request (c)

If the publisher can satisfy both changes, the publisher sends a positive result for the RPC. If the publisher cannot satisfy either of the proposed changes, the publisher sends an RPC error response (e). The following is an example RPC error response for (e) which includes a hint. This hint is an alternative time period value which might have resulted in a successful modification:

```
HTTP status code - 406

{ "ietf-restconf:errors" : {
    "error" : [
      "error-type": "application",
      "error-tag": "operation-failed",
      "error-severity": "error",
      "error-app-tag": "ietf-yang-push:period-unsupported",
      "error-info": {
        "ietf-yang-push":
        "modify-subscription-datastore-error-info": {
           "period-hint": "3000"
        }
      }
    ]
  }
}
```

Figure 11: Modify subscription failure with Hint (e)

B.1.3. Deleting Dynamic Subscriptions

The following demonstrates deleting a subscription. This subscription may have been to either a stream or a datastore.

```
POST /restconf/operations/subscriptions:delete-subscription

{
 "delete-subscription": {
    "identifier": "22"
 }
}
```

Figure 12: Delete subscription

If the publisher can satisfy the request, the publisher replies with success to the RPC request.

If the publisher cannot satisfy the request, the publisher sends an error-rpc element indicating the modification didn’t work. Figure 13 shows a valid response for existing valid subscription identifier, but that subscription identifier was created on a different transport session:

```
HTTP status code - 406

{
  "ietf-restconf:errors" : {
    "error" : [
      "error-type": "application",
      "error-tag": "operation-failed",
      "error-severity": "error",
      "error-app-tag":
         "ietf-subscribed-notifications:no-such-subscription"
    ]
  }
}
```

Figure 13: Unsuccessful delete subscription

B.2. Configured Subscriptions

Configured subscriptions may be established, modified, and deleted using configuration operations against the top-level subtree of [I-D.draft-ietf-netconf-subscribed-notifications] or [I-D.ietf-netconf-yang-push].

In this section, we present examples of how to manage the configuration subscriptions using a HTTP2 client.

B.2.1. Creating Configured Subscriptions

For subscription creation via configuration operations, a RESTCONF client may send:

```
POST /restconf/operations/subscriptions/

{
   "edit-config": {
      "target": {
         "running": null
      },
      "default-operation": "none",
      "config": {
         "subscriptions": {
            "subscription": {
               "identifier": "22",
               "transport": "HTTP2",
               "stream": "NETCONF",
               "receivers": {
                  "receiver": {
                     "name": "receiver1",
                     "address": "1.2.3.4"
                  }
               }
            }
         }
      }
   }
}
```

Figure 14: Create a configured subscription

If the request is accepted, the publisher will indicate this. If the request is not accepted because the publisher cannot serve it, no configuration is changed. In this case the publisher may reply:

HTTP status code - 406


```
{
  "ietf-restconf:errors" : {
    "error" : [
      "error-type": "application",
      "error-tag": "resource-denied",
      "error-severity": "error",
      "error-message": {
         "@lang": "en",
         "#text": "Temporarily the publisher cannot serve this
         subscription due to the current workload."
      }
    ]
  }
}
```


Figure 15: Response to a failed configured subscription establishment

After a subscription has been created and been verified as VALID, HTTP2 connectivity to each receiver will be established if that connectivity does not already exist.

The following figure shows the interaction model for the successful creation of a configured subscription.

```
    +----------+                 +-----------+     +---------+
    |Config Ops|                 | Publisher |     | 1.2.3.4 |
    +----------+                 +-----------+     +---------+
         |                            |                |
         |    Capability Exchange     |                |
         |<-------------------------->|                |
         |                            |                |
         |                            |                |
         |        Edit-config         |                |
         |--------------------------->|                |
         |       RPC Reply: OK        |                |
         |<---------------------------|                |
         |                            |   Call Home    |
         |                            |<-------------->|
         |                            |                |
         |                            |  subscription- |
         |                            |  started       |
         |                            |--------------->|
         |                            |                |
         |                            |  notification  |
         |                            |  message       |
         |                            |--------------->|
```

Figure 16: Interaction model for configured subscription establishment

B.2.2. Modifying Configured Subscriptions

Configured subscriptions can be modified using configuration operations against the top-level container "/subscriptions".

For example, the subscription established in the previous section could be modified as follows, here a adding a second receiver:

```
POST /restconf/operations/subscriptions

{
   "edit-config": {
      "target": {
         "running": null
      },
      "config": {
         "subscriptions": {
            "subscription": {
               "identifier": "1922",
               "receivers": {
                  "receiver": {
                     "name": "receiver2",
                     "address": "1.2.3.5"
                  }
               }
            }
         }
      }
   }
}
```

Figure 17: Modify configured subscription

If the request is accepted, the publisher will indicate success. The result is that the interaction model described in Figure 16 may be extended as follows.

```
 +----------+                 +-----------+     +---------+  +---------+
 |Config Ops|                 | Publisher |     | 1.2.3.4 |  | 1.2.3.5 |
 +----------+                 +-----------+     +---------+  +---------+
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |        Edit-config         |                |            |
       |--------------------------->|                |            |
       |       RPC Reply: OK        |                |            |
       |<---------------------------|                |            |
       |                            |  subscription- |            |
       |                            |  started       |            |
       |                            |----------------------------->|
       |                            |                |            |
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |                            |----------------------------->|
       |                            |                |            |
```

Figure 18: Interaction model for configured subscription modification

Note in the above that in the specific example above, modifying a configured subscription actually resulted in "subscription-started" notification. And because of existing HTTP2 connectivity, no additional call home was needed. Also note that if the edit of the configuration had impacted the filter, a separate modify-subscription would have been required for the original receiver.

B.2.3. Deleting Configured Subscriptions

Configured subscriptions can be deleted using configuration operations against the top-level container "/subscriptions". Deleting the subscription above would result in the following flow impacting all active receivers.

```
 +----------+                 +-----------+     +---------+  +---------+
 |Config Ops|                 | Publisher |     | 1.2.3.4 |  | 1.2.3.5 |
 +----------+                 +-----------+     +---------+  +---------+
       |                            |                |            |
       |                            |  notification  |            |
       |                            |  message       |            |
       |                            |--------------->|            |
       |                            |----------------------------->|
       |                            |                |            |
       |        Edit-config         |                |            |
       |--------------------------->|                |            |
       |       RPC Reply: OK        |                |            |
       |<---------------------------|                |            |
       |                            |  subscription- |            |
       |                            |  terminated    |            |
       |                            |--------------->|            |
       |                            |----------------------------->|
       |                            |                |            |
```

Figure 19: Interaction model for configured subscription deletion

B.3. Subscription State Notifications

A publisher will send subscription state notifications according to the definitions within [I-D.draft-ietf-netconf-subscribed-notifications]).

B.3.1. subscription-started and subscription-modified

A "subscription-started" encoded in JSON would look like:

```
{
  "ietf-restconf:notification" : {
    "eventTime": "2007-09-01T10:00:00Z",
    "ietf-subscribed-notifications:subscription-started": {
      "identifier": "39",
      "transport": "HTTP2",
      "stream-xpath-filter": "/ex:foo",
      "stream": {
         "ietf-netconf-subscribed-notifications" : "NETCONF"
      }
    }
  }
}
```

Figure 20: subscription-started subscription state notification

The "subscription-modified" is identical to Figure 20, with just the word "started" being replaced by "modified".

B.3.2. subscription-completed, subscription-resumed, and replay-complete

A "subscription-completed" would look like:

```
{
  "ietf-restconf:notification" : {
    "eventTime": "2007-09-01T10:00:00Z",
    "ietf-subscribed-notifications:subscription-completed": {
      "identifier": "39",
    }
  }
}
```

Figure 21: subscription-completed notification in JSON

The "subscription-resumed" and "replay-complete" are virtually identical, with "subscription-completed" simply being replaced by "subscription-resumed" and "replay-complete".

B.3.3. subscription-terminated and subscription-suspended

A "subscription-terminated" would look like:

```
{
  "ietf-restconf:notification" : {
    "eventTime": "2007-09-01T10:00:00Z",
    "ietf-subscribed-notifications:subscription-terminated": {
      "identifier": "39",
      "error-id": "suspension-timeout"
    }
  }
}
```

Figure 22: subscription-terminated subscription state notification

The "subscription-suspended" is virtually identical, with "subscription-terminated" simply being replaced by "subscription-suspended".

Appendix C. Changes between revisions

(To be removed by RFC editor prior to publication)

v05 - v06

- o JSON examples updated by Reshad.

v04 - v05

- o Error mechanisms updated to match embedded RESTCONF mechanisms
- o Restructured format and sections of document.
- o Added a YANG data model for HTTP specific parameters.
- o Mirrored the examples from the NETCONF transport draft to allow easy comparison.

v03 - v04

- o Draft not fully synched to new version of subscribed-notifications yet.
- o References updated

v02 - v03

- o Event notification reframed to notification message.
- o Tweaks to wording/capitalization/format.

v01 - v02

- o Removed sections now redundant with [I-D.draft-ietf-netconf-subscribed-notifications] and [I-D.ietf-netconf-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

Reshad Rahman
Cisco Systems

Email: rrahman@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 6, 2018

K. Watsen
Juniper Networks
G. Wu
Cisco Systems

June 4, 2018

YANG Groupings for SSH Clients and SSH Servers
draft-ietf-netconf-ssh-client-server-06

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:

- I-D.ietf-netconf-trust-anchors

- I-D.ietf-netconf-keystore

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

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

- "YYYY" --> the assigned RFC value for I-D.ietf-netconf-trust-anchors

- "ZZZZ" --> 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:

- "2018-06-04" --> 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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 4
3. The SSH Client Model 4
  3.1. Tree Diagram 4
  3.2. Example Usage 6
  3.3. YANG Module 9
4. The SSH Server Model 12
  4.1. Tree Diagram 12
  4.2. Example Usage 14
  4.3. YANG Module 17

5. The SSH Common Model 20
5.1. Tree Diagram 21
5.2. Example Usage 21
5.3. YANG Module 22
6. Security Considerations 32
7. IANA Considerations 33
7.1. The IETF XML Registry 33
7.2. The YANG Module Names Registry 33
8. Acknowledgements 34
9. References 34
9.1. Normative References 34
9.2. Informative References 35
Appendix A. Change Log 37
A.1. 00 to 01 37
A.2. 01 to 02 37
A.3. 02 to 03 37
A.4. 03 to 04 37
A.5. 04 to 05 38
A.6. 05 to 06 38
Authors' Addresses 38

1. Introduction

This document defines three YANG 1.1 [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. It is intended that these groupings will be used by applications using the SSH protocol [RFC4252], [RFC4253], and [RFC4254]. For instance, these groupings could be used to help define the data model for an OpenSSH [OPENSSH] 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 affords 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 "ssh-server-grouping" grouping for the SSH parts it provides, while adding data nodes for the TCP-level call-home configuration.

The modules defined in this document uses groupings defined in [I-D.ietf-netconf-keystore] enabling keys to be either locally defined or a reference to globally configured values.

The modules defined in this document optionally support [RFC6187] enabling X.509v3 certificate based host keys and public keys.

2. 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.

3. The SSH Client Model

3.1. Tree Diagram

This section provides two tree diagrams [RFC8340] for the "ietf-ssh-client" module, the first with used groupings expanded and the second with used groupings not expanded.

The following tree diagram has used groupings expanded:

```
module: ietf-ssh-client

  grouping ssh-client-grouping
    +-- client-identity
    |  +-- username?           string
    |  +-- (auth-type)
    |     +--:(password)
    |     |  +-- password?      string
    |     +--:(public-key)
    |     |  +-- public-key
    |     |     +-- (local-or-keystore)
    |     |        +--:(local)
    |     |        |  +-- algorithm      ct:key-algorithm-ref
    |     |        |  +-- public-key     binary
    |     |        |  +-- private-key    union
    |     |        +--:(keystore) {keystore-implemented}?
    |     |           +-- reference      ks:asymmetric-key-ref
    |     +--:(certificate)
    |        +-- certificate {sshcmn:ssh-x509-certs}?
    |           +-- (local-or-keystore)
    |              +--:(local)
    |              |  +-- algorithm
    |              |  |       ct:key-algorithm-ref
    |              |  +-- public-key                binary
    |              |  +-- private-key               union
    |              |  +-- cert
    |              |  |       ct:end-entity-cert-cms
    |              |  +---n certificate-expiration
    |              |     +-- expiration-date?   yang:date-and-time
    |              +--:(keystore) {keystore-implemented}?
    |                 +-- reference
    |                         ks:asymmetric-key-certificate-ref
    +-- server-auth
    |  +-- pinned-ssh-host-keys?   ta:pinned-host-keys-ref
    |  +-- pinned-ca-certs?        ta:pinned-certificates-ref
    |  |       {sshcmn:ssh-x509-certs}?
    |  +-- pinned-server-certs?    ta:pinned-certificates-ref
    |          {sshcmn:ssh-x509-certs}?
    +-- 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
```

The following tree diagram does not have the groupings expanded:

[Note: '\' line wrapping for formatting only]

```
module: ietf-ssh-client

  grouping ssh-client-grouping
    +-- client-identity
    |  +-- username?            string
    |  +-- (auth-type)
    |     +--:(password)
    |     |  +-- password?      string
    |     +--:(public-key)
    |     |  +-- public-key
    |     |     +---u ks:local-or-keystore-asymmetric-key-grouping
    |     +--:(certificate)
    |        +-- certificate {sshcmn:ssh-x509-certs}?
    |           +---u ks:local-or-keystore-end-entity-certificate-gr\
ouping
    +-- server-auth
    |  +-- pinned-ssh-host-keys?   ta:pinned-host-keys-ref
    |  +-- pinned-ca-certs?        ta:pinned-certificates-ref
    |  |       {sshcmn:ssh-x509-certs}?
    |  +-- pinned-server-certs?    ta:pinned-certificates-ref
    |          {sshcmn:ssh-x509-certs}?
    +-- transport-params {ssh-client-transport-params-config}?
       +---u sshcmn:transport-params-grouping
```

3.2. Example Usage

This section presents two examples showing the ssh-client-grouping populated with some data. These examples are effectively the same except the first configures the client identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 3 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following example configures the client identity using a local key:

[Note: '\' line wrapping for formatting only]

```
<ssh-client
   xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-client"
   xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common">

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <username>foobar</username>
    <public-key>
      <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-t\
ypes">ct:rsa1024</algorithm>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
    </public-key>
  </client-identity>

  <!-- which host-keys will this client trust -->
  <server-auth>
    <pinned-ssh-host-keys>explicitly-trusted-ssh-host-keys</pinned-s\
sh-host-keys>
  </server-auth>

  <transport-params>
    <host-key>
      <host-key-alg>algs:ssh-rsa</host-key-alg>
    </host-key>
    <key-exchange>
      <key-exchange-alg>
        algs:diffie-hellman-group-exchange-sha256
      </key-exchange-alg>
    </key-exchange>
    <encryption>
      <encryption-alg>algs:aes256-ctr</encryption-alg>
      <encryption-alg>algs:aes192-ctr</encryption-alg>
      <encryption-alg>algs:aes128-ctr</encryption-alg>
      <encryption-alg>algs:aes256-cbc</encryption-alg>
      <encryption-alg>algs:aes192-cbc</encryption-alg>
      <encryption-alg>algs:aes128-cbc</encryption-alg>
    </encryption>
    <mac>
      <mac-alg>algs:hmac-sha2-256</mac-alg>
      <mac-alg>algs:hmac-sha2-512</mac-alg>
    </mac>
  </transport-params>

</ssh-client>
```

The following example configures the client identity using a key from the keystore:

[Note: '\' line wrapping for formatting only]

```
<ssh-client
   xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-client"
   xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common">

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

  <!-- which host-keys will this client trust -->
  <server-auth>
    <pinned-ssh-host-keys>explicitly-trusted-ssh-host-keys</pinned-s\
sh-host-keys>
  </server-auth>

  <transport-params>
    <host-key>
      <host-key-alg>algs:ssh-rsa</host-key-alg>
    </host-key>
    <key-exchange>
      <key-exchange-alg>
        algs:diffie-hellman-group-exchange-sha256
      </key-exchange-alg>
    </key-exchange>
    <encryption>
      <encryption-alg>algs:aes256-ctr</encryption-alg>
      <encryption-alg>algs:aes192-ctr</encryption-alg>
      <encryption-alg>algs:aes128-ctr</encryption-alg>
      <encryption-alg>algs:aes256-cbc</encryption-alg>
      <encryption-alg>algs:aes192-cbc</encryption-alg>
      <encryption-alg>algs:aes128-cbc</encryption-alg>
    </encryption>
    <mac>
      <mac-alg>algs:hmac-sha2-256</mac-alg>
      <mac-alg>algs:hmac-sha2-512</mac-alg>
    </mac>
  </transport-params>

</ssh-client>
```

3.3. YANG Module

This YANG module has normative references to [RFC6536], [I-D.ietf-netconf-trust-anchors], and [I-D.ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-ssh-client@2018-06-04.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 sshcmn;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
  }

  import ietf-trust-anchors {
    prefix ta;
    reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC ZZZZ:
         YANG Data Model for a Centralized Keystore Mechanism";
  }

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

  contact
   "WG Web:   <http://datatracker.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) 2018 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 "2018-06-04" {
       description
        "Initial version";
       reference
        "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
     }

     // features

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

     // groupings

     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 client-identity {
         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 {
            mandatory true;
            description
              "The authentication type.";
            leaf password {
              type string;
              description
                "A password to be used for client authentication.";
            }
            container public-key {
              uses ks:local-or-keystore-asymmetric-key-grouping;
              description
                "A locally-defined or referenced asymmetric key pair
                 to be used for client authentication.";
              reference
                "RFC ZZZZ:
                  YANG Data Model for a Centralized Keystore Mechanism";
            }
            container certificate {
              if-feature sshcmn:ssh-x509-certs;
              uses ks:local-or-keystore-end-entity-certificate-grouping;
              description
                "A locally-defined or referenced certificate
                 to be used for client authentication.";
              reference
                "RFC ZZZZ
                  YANG Data Model for a Centralized Keystore Mechanism";
            }
          } // end auth-type
        } // end client-identity

        container server-auth {
          must 'pinned-ssh-host-keys or pinned-ca-certs or '
               + 'pinned-server-certs';
          description
            "Trusted server identities.";
          leaf pinned-ssh-host-keys {
            type ta:pinned-host-keys-ref;
            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 authenticated if it is an exact
               match to a configured SSH host key.";
            reference
              "RFC YYYY: YANG Data Model for Global Trust Anchors";
          }
```

```
          leaf pinned-ca-certs {
            if-feature sshcmn:ssh-x509-certs;
            type ta:pinned-certificates-ref;
            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 CA certificate.";
            reference
              "RFC YYYY: YANG Data Model for Global Trust Anchors";
          }

          leaf pinned-server-certs {
            if-feature sshcmn:ssh-x509-certs;
            type ta:pinned-certificates-ref;
            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 server certificate.";
            reference
              "RFC YYYY: YANG Data Model for Global Trust Anchors";
          }
        } // end server-auth

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

      } // end ssh-client-grouping

    }
    <CODE ENDS>
```

4. The SSH Server Model

4.1. Tree Diagram

This section provides two tree diagrams [RFC8340] for the "ietf-ssh-server" module, the first with used groupings expanded and the second with used groupings not expanded.

The following tree diagram has used groupings expanded:

```
   [Note: '\' line wrapping for formatting only]

   module: ietf-ssh-server

     grouping ssh-server-grouping
       +-- server-identity
       |  +-- host-key* [name]
       |     +-- name?                string
       |     +-- (host-key-type)
       |        +--:(public-key)
       |        |  +-- public-key
       |        |     +-- (local-or-keystore)
       |        |        +--:(local)
       |        |        |  +-- algorithm      ct:key-algorithm-ref
       |        |        |  +-- public-key     binary
       |        |        |  +-- private-key    union
       |        |        +--:(keystore) {keystore-implemented}?
       |        |           +-- reference      ks:asymmetric-key-ref
       |        +--:(certificate)
       |           +-- certificate {sshcmn:ssh-x509-certs}?
       |              +-- (local-or-keystore)
       |                 +--:(local)
       |                 |  +-- algorithm
       |                 |  |       ct:key-algorithm-ref
       |                 |  +-- public-key                binary
       |                 |  +-- private-key               union
       |                 |  +-- cert
       |                 |  |       ct:end-entity-cert-cms
       |                 |  +---n certificate-expiration
       |                 |     +-- expiration-date?   yang:date-and-tim\
   e
       |                 +--:(keystore) {keystore-implemented}?
       |                    +-- reference
       |                            ks:asymmetric-key-certificate-ref
       +-- client-cert-auth {sshcmn:ssh-x509-certs}?
       |  +-- pinned-ca-certs?       ta:pinned-certificates-ref
       |  +-- pinned-client-certs?   ta:pinned-certificates-ref
       +-- 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
```

The following tree diagram does not have the used groupings expanded:

```
[Note: '\' line wrapping for formatting only]

module: ietf-ssh-server

  grouping ssh-server-grouping
    +-- server-identity
    |  +-- host-key* [name]
    |     +-- name?                string
    |     +-- (host-key-type)
    |        +--:(public-key)
    |        |  +-- public-key
    |        |     +---u ks:local-or-keystore-asymmetric-key-groupin\
g
    |        +--:(certificate)
    |           +-- certificate {sshcmn:ssh-x509-certs}?
    |              +---u ks:local-or-keystore-end-entity-certificate\
-grouping
    +-- client-cert-auth {sshcmn:ssh-x509-certs}?
    |  +-- pinned-ca-certs?       ta:pinned-certificates-ref
    |  +-- pinned-client-certs?   ta:pinned-certificates-ref
    +-- transport-params {ssh-server-transport-params-config}?
       +---u sshcmn:transport-params-grouping
```

4.2. Example Usage

This section presents two examples showing the ssh-server-grouping populated with some data. These examples are effectively the same except the first configures the server identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 3 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following example configures the server identity using a local key:

```
[Note: '\' line wrapping for formatting only]

<ssh-server xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server"
            xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common"\
>

  <!-- which host-keys will this SSH server present -->
  <server-identity>
    <host-key>
      <name>deployment-specific-certificate</name>
      <public-key>
```

```
            <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto\
   -types">ct:rsa1024</algorithm>
            <private-key>base64encodedvalue==</private-key>
            <public-key>base64encodedvalue==</public-key>
          </public-key>
        </host-key>
      </server-identity>

      <!-- which client-certs will this SSH server trust -->
      <client-cert-auth>
        <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-c\
   erts>
        <pinned-client-certs>explicitly-trusted-client-certs</pinned-cli\
   ent-certs>
      </client-cert-auth>

      <transport-params>
        <host-key>
          <host-key-alg>algs:ssh-rsa</host-key-alg>
        </host-key>
        <key-exchange>
          <key-exchange-alg>
            algs:diffie-hellman-group-exchange-sha256
          </key-exchange-alg>
        </key-exchange>
        <encryption>
          <encryption-alg>algs:aes256-ctr</encryption-alg>
          <encryption-alg>algs:aes192-ctr</encryption-alg>
          <encryption-alg>algs:aes128-ctr</encryption-alg>
          <encryption-alg>algs:aes256-cbc</encryption-alg>
          <encryption-alg>algs:aes192-cbc</encryption-alg>
          <encryption-alg>algs:aes128-cbc</encryption-alg>
        </encryption>
        <mac>
          <mac-alg>algs:hmac-sha2-256</mac-alg>
          <mac-alg>algs:hmac-sha2-512</mac-alg>
        </mac>
      </transport-params>

   </ssh-server>
```

The following example configures the server identity using a key from the keystore:

[Note: '\' line wrapping for formatting only]

```
<ssh-server xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-server"
            xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common"\
>

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

  <!-- which client-certs will this SSH server trust -->
  <client-cert-auth>
    <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-c\
erts>
    <pinned-client-certs>explicitly-trusted-client-certs</pinned-cli\
ent-certs>
  </client-cert-auth>

  <transport-params>
    <host-key>
      <host-key-alg>algs:ssh-rsa</host-key-alg>
    </host-key>
    <key-exchange>
      <key-exchange-alg>
        algs:diffie-hellman-group-exchange-sha256
      </key-exchange-alg>
    </key-exchange>
    <encryption>
      <encryption-alg>algs:aes256-ctr</encryption-alg>
      <encryption-alg>algs:aes192-ctr</encryption-alg>
      <encryption-alg>algs:aes128-ctr</encryption-alg>
      <encryption-alg>algs:aes256-cbc</encryption-alg>
      <encryption-alg>algs:aes192-cbc</encryption-alg>
      <encryption-alg>algs:aes128-cbc</encryption-alg>
    </encryption>
    <mac>
      <mac-alg>algs:hmac-sha2-256</mac-alg>
      <mac-alg>algs:hmac-sha2-512</mac-alg>
    </mac>
  </transport-params>

</ssh-server>
```

4.3. YANG Module

This YANG module has normative references to [I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore] and informative references to [RFC4253] and [RFC7317].

```
<CODE BEGINS> file "ietf-ssh-server@2018-06-04.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 sshcmn;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
  }

  import ietf-trust-anchors {
    prefix ta;
    reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC ZZZZ:
         YANG Data Model for a Centralized Keystore Mechanism";
  }

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

  contact
   "WG Web:   <http://datatracker.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 server that
       can be used as a basis for specific SSH server instances.

       Copyright (c) 2018 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 "2018-06-04" {
       description
        "Initial version";
       reference
        "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
     }

     // features

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

     // groupings

     grouping ssh-server-grouping {
       description
         "A reusable grouping for configuring a SSH server without
          any consideration for how underlying TCP sessions are
          established.";

       container server-identity {
         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";
             container public-key {
               uses ks:local-or-keystore-asymmetric-key-grouping;
               description
                 "A locally-defined or referenced asymmetric key pair
                  to be used for the SSH server's host key.";
               reference
                 "RFC ZZZZ: YANG Data Model for a Centralized
                            Keystore Mechanism";
             }
             container certificate {
               if-feature sshcmn:ssh-x509-certs;
               uses
                 ks:local-or-keystore-end-entity-certificate-grouping;
               description
                 "A locally-defined or referenced end-entity
                  certificate to be used for the SSH server's
                  host key.";
               reference
                 "RFC ZZZZ: YANG Data Model for a Centralized
                            Keystore Mechanism";
             }
           }
         }
       }

       container client-cert-auth {
         if-feature sshcmn:ssh-x509-certs;
         description
           "A reference to a list of pinned certificate authority (CA)
            certificates and a reference to a list of pinned client
            certificates.
```

```
            Note: password and public-key based client authentication
                  is not configured in this YANG module as they are
                  expected to be configured by the ietf-system module
                  defined in RFC 7317 ";
         reference
           "RFC 7317: A YANG Data Model for System Management";
         leaf pinned-ca-certs {
           type ta:pinned-certificates-ref;
           description
             "A reference to a list of certificate authority (CA)
              certificates used by the SSH server to authenticate
              SSH client certificates.  A client certificate is
              authenticated if it has a valid chain of trust to
              a configured pinned CA certificate.";
           reference
             "RFC YYYY: YANG Data Model for Global Trust Anchors";
         }
         leaf pinned-client-certs {
           type ta:pinned-certificates-ref;
           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 pinned client certificate.";
           reference
             "RFC YYYY: YANG Data Model for Global Trust Anchors";
         }
       }

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

     } // end ssh-server-grouping

   }
   <CODE ENDS>
```

5. 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.

5.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-ssh-common" module.

```
module: ietf-ssh-common

  grouping transport-params-grouping
    +-- host-key
    |  +-- host-key-alg*   identityref
    +-- key-exchange
    |  +-- key-exchange-alg*   identityref
    +-- encryption
    |  +-- encryption-alg*   identityref
    +-- mac
       +-- mac-alg*   identityref
```

5.2. Example Usage

This following example illustrates how the transport-params-grouping appears when populated with some data.

```
<transport-params
  xmlns="urn:ietf:params:xml:ns:yang:ietf-ssh-common"
  xmlns:algs="urn:ietf:params:xml:ns:yang:ietf-ssh-common">
  <host-key>
    <host-key-alg>algs:x509v3-rsa2048-sha256</host-key-alg>
    <host-key-alg>algs:ssh-rsa</host-key-alg>
  </host-key>
  <key-exchange>
    <key-exchange-alg>
      algs:diffie-hellman-group-exchange-sha256
    </key-exchange-alg>
  </key-exchange>
  <encryption>
    <encryption-alg>algs:aes256-ctr</encryption-alg>
    <encryption-alg>algs:aes192-ctr</encryption-alg>
    <encryption-alg>algs:aes128-ctr</encryption-alg>
    <encryption-alg>algs:aes256-cbc</encryption-alg>
    <encryption-alg>algs:aes192-cbc</encryption-alg>
    <encryption-alg>algs:aes128-cbc</encryption-alg>
  </encryption>
  <mac>
    <mac-alg>algs:hmac-sha2-256</mac-alg>
    <mac-alg>algs:hmac-sha2-512</mac-alg>
  </mac>
</transport-params>
```

5.3. YANG Module

This YANG module has normative references to [RFC4253], [RFC4344], [RFC4419], [RFC5656], [RFC6187], and [RFC6668].

```
<CODE BEGINS> file "ietf-ssh-common@2018-06-04.yang"
module ietf-ssh-common {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ssh-common";
  prefix "sshcmn";

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

  contact
   "WG Web:   <http://datatracker.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) 2018 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 "2018-06-04" {
       description
        "Initial version";
       reference
        "RFC XXXX: YANG Groupings for SSH Clients and SSH Servers";
     }

     // 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 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)";
    }

    // 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";
    }

    // groupings

    grouping transport-params-grouping {
      description
        "A reusable grouping for SSH transport parameters.";
      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
            "Acceptable host key algorithms in order of descending
             preference.

             If this leaf-list is not configured (has zero elements)
             the acceptable host key algorithms are implementation-
             defined.";
        }
      }
```

```
        container key-exchange {
          description
            "Parameters regarding key exchange.";
          leaf-list key-exchange-alg {
            type identityref {
              base key-exchange-alg-base;
            }
            ordered-by user;
            description
             "Acceptable key exchange algorithms in order of descending
              preference.

              If this leaf-list is not configured (has zero elements)
              the acceptable key exchange algorithms are implementation
              defined.";
          }
        }
        container encryption {
          description
            "Parameters regarding encryption.";
          leaf-list encryption-alg {
            type identityref {
              base encryption-alg-base;
            }
            ordered-by user;
            description
             "Acceptable encryption algorithms in order of descending
              preference.

              If this leaf-list is not configured (has zero elements)
              the acceptable encryption algorithms are implementation
              defined.";
          }
        }
        container mac {
          description
            "Parameters regarding message authentication code (MAC).";
          leaf-list mac-alg {
            type identityref {
              base mac-alg-base;
            }
            ordered-by user;
            description
              "Acceptable MAC algorithms in order of descending
               preference.

               If this leaf-list is not configured (has zero elements)
               the acceptable MAC algorithms are implementation-
```

```
              defined.";
          }
        }

      } // transport-params-grouping

    }
    <CODE ENDS>
```

6. Security Considerations

The YANG modules defined in this document are 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.

Since the modules defined in this document define only groupings, these considerations are primarily for the designers of other modules that use these groupings.

There are a number of data nodes defined in the YANG modules 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 all the modules defined in this draft are 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 the YANG modules 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 in the 'ietf-ssh-client' module is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. The only time this node should be returned is to support backup/restore type workflows. However, no NACM annotations are applied as the data SHOULD be writable by users other than a designated 'recovery session'.

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

7. IANA Considerations

7.1. 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.

7.2. 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:       sshcmn
reference:    RFC XXXX
```

8. 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.

9. References

9.1. Normative References

[I-D.ietf-netconf-keystore]
Watsen, K., "YANG Data Model for a "Keystore" Mechanism", draft-ietf-netconf-keystore-04 (work in progress), October 2017.

[I-D.ietf-netconf-trust-anchors]
Watsen, K., "YANG Data Model for Global Trust Anchors", draft-ietf-netconf-trust-anchors-00 (work in progress), June 2018.

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

[RFC4344] Bellare, M., Kohno, T., and C. Namprempre, "The Secure Shell (SSH) Transport Layer Encryption Modes", RFC 4344, DOI 10.17487/RFC4344, January 2006, <https://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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6668>.

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

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

9.2. Informative References

[OPENSSH] "OpenSSH", 2016, <http://www.openssh.com>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Change Log

A.1. 00 to 01

- Noted that ’0.0.0.0’ and ’::’ might have special meanings.
- Renamed "keychain" to "keystore".

A.2. 01 to 02

- Removed the groupings ’listening-ssh-client-grouping’ and ’listening-ssh-server-grouping’. Now modules only contain the transport-independent groupings.
- Simplified the "client-auth" part in the ietf-ssh-client module. It now inlines what it used to point to keystore for.
- Added cipher suites for various algorithms into new ’ietf-ssh-common’ module.

A.3. 02 to 03

- Removed ’RESTRICTED’ enum from ’password’ leaf type.
- Added a ’must’ statement to container ’server-auth’ asserting that at least one of the various auth mechanisms must be specified.
- Fixed description statement for leaf ’trusted-ca-certs’.

A.4. 03 to 04

- Change title to "YANG Groupings for SSH Clients and SSH Servers"
- Added reference to RFC 6668
- Added RFC 8174 to Requirements Language Section.
- Enhanced description statement for ietf-ssh-server’s "trusted-ca-certs" leaf.
- Added mandatory true to ietf-ssh-client’s "client-auth" ’choice’ statement.
- Changed the YANG prefix for module ietf-ssh-common from ’sshcom’ to ’sshcmn’.
- Removed the compression algorithms as they are not commonly configurable in vendors’ implementations.

- o Updating descriptions in transport-params-grouping and the servers's usage of it.

- o Now tree diagrams reference ietf-netmod-yang-tree-diagrams

- o Updated YANG to use typedefs around leafrefs to common keystore paths

- o Now inlines key and certificates (no longer a leafref to keystore)

A.5. 04 to 05

- o Merged changes from co-author.

A.6. 05 to 06

- o Updated to use trust anchors from trust-anchors draft (was keystore draft)

- o Now uses new keystore grouping enabling asymmetric key to be either locally defined or a reference to the keystore.

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, 2019

E. Voit
Cisco Systems
A. Clemm
Huawei
A. Gonzalez Prieto
VMWare
E. Nilsen-Nygaard
A. Tripathy
Cisco Systems
July 2, 2018

Customized Subscriptions to a Publisher's Event Streams
draft-ietf-netconf-subscribed-notifications-14

Abstract

This document defines a YANG data model and associated mechanisms enabling subscriber-specific subscriptions to a publisher's event streams. Applying these elements allows a subscriber to request for and receive a continuous, custom feed 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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 3
  1.2. Terminology 3
  1.3. Solution Overview 5
  1.4. Relationship to RFC-5277 6
2. Solution 6
  2.1. Event Streams 7
  2.2. Event Stream Filters 7
  2.3. QoS 8
  2.4. Dynamic Subscriptions 9
  2.5. Configured Subscriptions 16
  2.6. Event Record Delivery 23
  2.7. Subscription State Notifications 24
  2.8. Subscription Monitoring 30
  2.9. Advertisement 31
3. YANG Data Model Trees 31
  3.1. Event Streams Container 31
  3.2. Filters Container 32
  3.3. Subscriptions Container 32
4. Data Model 34
5. Considerations 61
  5.1. IANA Considerations 61
  5.2. Implementation Considerations 61
  5.3. Transport Requirements 62
  5.4. Security Considerations 63
6. Acknowledgments 66
7. References 66
  7.1. Normative References 66
  7.2. Informative References 68
Appendix A. Changes between revisions 69
Authors' Addresses 73

1. Introduction

This document defines a YANG data model and associated mechanisms enabling subscriber-specific subscriptions to a publisher's event streams. Effectively this enables a 'subscribe then publish' capability where the customized information needs and access permissions of each target receiver are understood by the publisher

before subscribed event records are marshaled and pushed. The receiver then gets a continuous, custom feed of publisher generated information.

While the functionality defined in this document is transport-agnostic, transports like NETCONF [RFC6241] or RESTCONF [RFC8040] can be used to configure or dynamically signal subscriptions, and there are bindings defined for subscribed event record delivery for NETCONF within [I-D.draft-ietf-netconf-netconf-event-notifications], and for HTTP2 or HTTP1.1 within [I-D.draft-ietf-netconf-restconf-notif].

The YANG model in this document conforms to the Network Management Datastore Architecture defined in [RFC8342].

1.1. Motivation

Various limitations in [RFC5277] are discussed in [RFC7923]. Resolving these issues is the primary motivation for this work. Key capabilities supported by this document include:

- o multiple subscriptions on a single transport session
- o support for dynamic and configured subscriptions
- o modification of an existing subscription in progress
- o per-subscription operational counters
- o negotiation of subscription parameters (through the use of hints returned as part of declined subscription requests)
- o subscription state change notifications (e.g., publisher driven suspension, parameter modification)
- o independence from transport

1.2. 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.

Client: defined in [RFC8342].

Configuration: defined in [RFC8342].

Configuration datastore: defined in [RFC8342].

Configured subscription: A subscription installed via configuration into a configuration datastore.

Dynamic subscription: A subscription created dynamically by a subscriber via a remote procedure call.

Event: An occurrence of something that may be of interest. Examples include a configuration change, a fault, a change in status, crossing a threshold, or an external input to the system.

Event occurrence time: a timestamp matching the time an originating process identified as when an event happened.

Event record: A set of information detailing an event.

Event stream: A continuous, chronologically ordered set of events aggregated under some context.

Event stream filter: Evaluation criteria which may be applied against event records within an event stream. Event records pass the filter when specified criteria are met.

Notification message: Information intended for a receiver indicating that one or more event(s) have occurred.

Publisher: An entity responsible for streaming notification messages per the terms of a subscription.

Receiver: A target to which a publisher pushes subscribed event records. For dynamic subscriptions, the receiver and subscriber are the same entity.

Subscriber: A client able to request and negotiate a contract for the generation and push of event records from a publisher. For dynamic subscriptions, the receiver and subscriber are the same entity.

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.

All YANG tree diagrams used in this document follow the notation defined in [RFC8340].

1.3. Solution Overview

This document describes a transport agnostic mechanism for subscribing to and receiving content from an event stream within a publisher. This mechanism is through the use of a subscription.

Two types of subscriptions are supported:

1. Dynamic subscriptions, where a subscriber initiates a subscription negotiation with a publisher via an RPC. If the publisher is able to serve this request, it accepts it, and then starts pushing notification messages back to the subscriber. If the publisher is not able to serve it as requested, then an error response is returned. This response MAY include hints at subscription parameters that, had they been present, would have enabled the dynamic subscription request to be accepted.

2. Configured subscriptions, which allow the management of subscriptions via a configuration so that a publisher can send notification messages to a receiver of a configured subscription. Support for configured subscriptions is optional, with its availability advertised via a YANG feature.

Additional characteristics differentiating configured from dynamic subscriptions include:

- The lifetime of a dynamic subscription is bound by the transport session used to establish it. For connection-oriented stateful transports like NETCONF, the loss of the transport session will result in the immediate termination of any associated dynamic subscriptions. For connectionless or stateless transports like HTTP, a lack of receipt acknowledgment of a sequential set of notification messages and/or keep-alives can be used to trigger a termination of a dynamic subscription. Contrast this to the lifetime of a configured subscription. This lifetime is driven by relevant configuration being present within the publisher’s applied configuration. Being tied to configuration operations implies configured subscriptions can be configured to persist across reboots, and implies a configured subscription can persist even when its publisher is fully disconnected from any network.

- 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 by the original subscriber, or a change to configuration data referenced by the subscription.

Note that there is no mixing-and-matching of dynamic and configured operations on a single subscription. Specifically, a configured subscription cannot be modified or deleted using RPCs defined in this document. Similarly, a subscription established via RPC cannot be modified through configuration operations. Also note that transport specific transport drafts based on this specification MUST detail the life cycles of both dynamic and configured subscriptions.

A publisher MAY terminate a dynamic subscription at any time. Similarly, it MAY decide to temporarily suspend the sending of notification messages for any dynamic subscription, or for one or more receivers of a configured subscription. Such termination or suspension is driven by internal considerations of the publisher.

1.4. Relationship to RFC-5277

This document is intended to provide a superset of the subscription capabilities initially defined within [RFC5277]. Especially when extending an existing [RFC5277] implementation, it is important to understand what has been reused and what has been replaced. Key relationships between these two documents include:

- o this document defines a transport independent capability, [RFC5277] is specific to NETCONF.
- o the data model in this document is used instead of the data model in Section 3.4 of [RFC5277] for the new operations.
- o the RPC operations in this draft replaces the operation "create-subscription" defined in [RFC5277], section 4.
- o the <notification> message of [RFC5277], Section 4 is used.
- o the included contents of the "NETCONF" event stream are identical between this document and [RFC5277].
- o a publisher MAY implement both the Notification Management Schema and RPCs defined in [RFC5277] and this new document concurrently.
- o unlike [RFC5277], this document enables a single transport session to intermix of notification messages and RPCs for different subscriptions.

2. Solution

Per the overview provided in Section 1.3, this section details the overall context, state machines, and subsystems which may be assembled to allow the subscription of events from a publisher.

2.1. Event Streams

An event stream is a named entity on a publisher which exposes a continuously updating set of event records. Each event stream is available for subscription. It is out of the scope of this document to identify a) how streams are defined (other than the NETCONF stream), b) how event records are defined/generated, and c) how event records are assigned to streams.

There is only one reserved event stream name within this document: "NETCONF". The "NETCONF" event stream contains all NETCONF XML event record information supported by the publisher, except for the subscription state notifications described in Section 2.7. Among these included NETCONF XML event records are individual YANG 1.1 notifications described in section 7.16 of [RFC7950]. Each of these YANG 1.1 notifications will be treated as a distinct event record. Beyond the "NETCONF" stream, implementations MAY define additional event streams.

As event records are created by a system, they may be assigned to one or more streams. The event record is distributed to a subscription's receiver(s) where: (1) a subscription includes the identified stream, and (2) subscription filtering does not exclude the event record from that receiver.

Access control permissions may be used to silently exclude event records from within an event stream for which the receiver has no read access. As an example of how this might be accomplished, see [RFC8341] section 3.4.6. Note that per Section 2.7 of this document, subscription state change notifications are never filtered out.

If no access control permissions are in place for event records on an event stream, then a receiver MUST be allowed access to all the event records. If subscriber permissions change during the lifecycle of a subscription and event stream access is no longer permitted, then the subscription MUST be terminated.

Event records MUST NOT be delivered to a receiver in a different order than they were placed onto an event stream.

2.2. Event Stream Filters

This document defines an extensible filtering mechanism. The filter itself is a boolean test which is placed on the content of an event record. A 'false' filtering result causes the event message to be excluded from delivery to a receiver. A filter never results in information being stripped from within an event record prior to that event record being encapsulated within a notification message. The

two optional event stream filtering syntaxes supported are [XPATH] and subtree [RFC6241].

If no event stream filter is provided within a subscription, all event records on an event stream are to be sent.

2.3. QoS

This document provide for several QoS parameters. These parameters indicate the treatment of a subscription relative to other traffic between publisher and receiver. Included are:

- o A "dscp" marking to differentiate prioritization of notification messages during network transit.

- o A "weighting" so that bandwidth proportional to this weighting can be allocated to this subscription relative to other subscriptions destined for that receiver.

- o a "dependency" upon another subscription.

If the publisher supports the "dscp" feature, then a subscription with a "dscp" leaf MUST result in a corresponding [RFC2474] DSCP marking being placed within the IP header of any resulting notification messages and subscription state change notifications.

For the "weighting" parameter, when concurrently dequeuing notification messages from multiple subscriptions to a receiver, the publisher MUST allocate bandwidth to each subscription proportionally to the weights assigned to those subscriptions. "Weighting" is an optional capability of the publisher; support for it is identified via the "qos" feature.

If a subscription has the "dependency" parameter set, then any buffered notification messages containing event records selected by the parent subscription MUST be dequeued prior to the notification messages of the dependent subscription. If notification messages have dependencies on each other, the notification message queued the longest MUST go first. If a "dependency" included within an RPC references a subscription which does not exist or is no longer accessible to that subscriber, that "dependency" MUST be silently removed. "Dependency" is an optional capability of the publisher; support for it is identified via the "qos" feature.

2.4. Dynamic Subscriptions

Dynamic subscriptions are managed via protocol operations (in the form of [RFC7950], Section 7.14 RPCs) made against targets located within the publisher. These RPCs have been designed extensibly so that they may be augmented for subscription targets beyond event streams. For examples of such augmentations, see the RPC augmentations within [I-D.ietf-netconf-yang-push]'s YANG model.

2.4.1. Dynamic Subscription State Model

Below is the publisher's state machine for a dynamic subscription. Each state is shown in its own box. It is important to note that such a subscription doesn't exist at the publisher until an "establish-subscription" RPC is accepted. The mere request by a subscriber to establish a subscription is insufficient for that subscription to be externally visible. Start and end states are depicted to reflect subscription creation and deletion events.

```
                    .........
                    : start :
                    :.......:
                        |
                establish-subscription
                        |
                        |   .-------modify-subscription--------.
                        v   v                                  |
                  .-----------.                          .-----------.
        .--------. | receiver  |--insufficient CPU, b/w-->| receiver  |
  modify-       '|  active   |                          | suspended |
  subscription   |           |<----CPU, b/w sufficient--|           |
      ---------->'-----------'                          '-----------'
                        |                                      |
             delete/kill-subscription                     delete/kill-
                        |                                 subscription
                        v                                      |
                    .........                                  |
                    :  end  :<---------------------------------'
                    :.......:
```

Figure 1: Publisher's state for a dynamic subscription

Of interest in this state machine are the following:

o Successful "establish-subscription" or "modify-subscription" RPCs put the subscription into the active state.

- o Failed "modify-subscription" 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, as will the reaching of a "stop-time".

- o A publisher may choose to suspend a subscription when there is insufficient CPU or bandwidth available to service the subscription. This is notified to a subscriber with a "subscription-suspended" state change notification.

- o A suspended subscription may be modified by the subscriber (for example in an attempt to use fewer resources). Successful modification returns the subscription to an active state.

- o Even without a "modify-subscription" request, a publisher may return a subscription to the active state should the resource constraints become sufficient again. This is announced to the subscriber via the "subscription-resumed" subscription state change notification.

2.4.2. Establishing a Dynamic Subscription

The "establish-subscription" RPC allows a subscriber to request the creation of a subscription. The transport selected by the subscriber to reach the publisher MUST be able to support multiple "establish-subscription" requests made within the same transport session.

The input parameters of the operation are:

- o A "stream" name which identifies the targeted event stream against which the subscription is applied.

- o An event stream filter which may reduce the set of event records pushed.

- o Where the transport used by the RPC supports multiple encodings, an optional "encoding" for the event records pushed. Note: If no "encoding" is included, the encoding of the RPC MUST be used.

- o An optional "stop-time" for the subscription. If no "stop-time" is present, notification messages will continue to be sent until the subscription is terminated.

- o An optional "start-time" for the subscription. The "start-time" MUST be in the past and indicates that the subscription is requesting a replay of previously generated information from the

event stream. For more on replay, see Section 2.4.2.1. Where there is no "start-time", the subscription starts immediately.

If the publisher can satisfy the "establish-subscription" request, it replies with an identifier for the subscription, and then immediately starts streaming notification messages.

Below is a tree diagram for "establish-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---x establish-subscription
   +---w input
   |  +---w (target)
   |  |  +--:(stream)
   |  |     +---w (stream-filter)?
   |  |     |  +--:(by-reference)
   |  |     |  |  +---w stream-filter-ref
   |  |     |  |          stream-filter-ref
   |  |     |  +--:(within-subscription)
   |  |     |     +---w (filter-spec)?
   |  |     |        +--:(stream-subtree-filter)
   |  |     |        |  +---w stream-subtree-filter?   <anydata>
   |  |     |        |          {subtree}?
   |  |     |        +--:(stream-xpath-filter)
   |  |     |           +---w stream-xpath-filter?
   |  |     |                   yang:xpath1.0 {xpath}?
   |  |     +---w stream                  stream-ref
   |  |     +---w replay-start-time?       yang:date-and-time
   |  |             {replay}?
   |  +---w stop-time?               yang:date-and-time
   |  +---w dscp?                    inet:dscp {dscp}?
   |  +---w weighting?               uint8 {qos}?
   |  +---w dependency?              subscription-id {qos}?
   |  +---w encoding?                encoding
   +--ro output
      +--ro identifier                     subscription-id
      +--ro replay-start-time-revision?   yang:date-and-time
              {replay}?
```

Figure 2: establish-subscription RPC tree diagram

A publisher MAY reject the "establish-subscription" RPC for many reasons as described in Section 2.4.6. The contents of the resulting RPC error response MAY include details on input parameters which if considered in a subsequent "establish-subscription" RPC, may result in a successful subscription establishment. Any such hints MUST be

transported within a yang-data "establish-subscription-stream-error-info" container included within the RPC error response.

```
yang-data establish-subscription-stream-error-info
   +--ro establish-subscription-stream-error-info
      +--ro reason?                   identityref
      +--ro filter-failure-hint?      string
```

Figure 3: establish-subscription RPC yang-data tree diagram

2.4.2.1. Requesting a replay of event records

Replay provides the ability to establish a subscription which is also capable of passing recently generated event records. In other words, as the subscription initializes itself, it sends any previously generated content from within the target event stream which meets the filter and timeframe criteria. The end of these historical event records is identified via a "replay-completed" state change notification. Any event records generated since the subscription establishment may then follow. For a particular subscription, all event records will be delivered in the order they are placed into the stream.

Replay is an optional feature which is dependent on an event stream supporting some form of logging. This document puts no restrictions on the size or form of the log, where it resides within the publisher, or when event record entries in the log are purged.

The inclusion of a "replay-start-time" within an "establish-subscription" RPC indicates a replay request. If the "replay-start-time" contains a value that is earlier than what a publisher’s retained history supports, then if the subscription is accepted, the actual publisher’s revised start time MUST be set in the returned "replay-start-time-revision" object.

A "stop-time" parameter may be included in a replay subscription. For a replay subscription, the "stop-time" MAY be earlier than the current time, but MUST be later than the "replay-start-time".

If the time the replay starts is later than the time marked within any event records retained within the replay buffer, then the publisher MUST send a "replay-completed" notification immediately after a successful establish-subscription RPC response.

If an event stream supports replay, the "replay-support" leaf is present in the "/streams/stream" list entry for the stream. An event stream that does support replay is not expected to have an unlimited supply of saved notifications available to accommodate any given

replay request. To assess the timeframe available for replay, subscribers can read the leafs "replay-log-creation-time" and "replay-log-aged-time". See Figure 18 for the YANG tree, and Section 4 for the YANG model describing these elements. The actual size of the replay log at any given time is a publisher specific matter. Control parameters for the replay log are outside the scope of this document.

2.4.3. Modifying a Dynamic Subscription

The "modify-subscription" operation permits changing the terms of an existing dynamic subscription. Dynamic subscriptions can be modified any number of times. If the publisher accepts the requested modifications, it acknowledges success to the subscriber, then immediately starts sending event records based on the new terms.

Subscriptions created by configuration cannot be modified via this RPC. However configuration may be used to modify objects referenced by the subscription (such as a referenced filter).

Below is a tree diagram for "modify-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---x modify-subscription
   +---w input
      +---w identifier               subscription-id
      +---w (target)
      |  +--:(stream)
      |     +---w (stream-filter)?
      |        +--:(by-reference)
      |        |  +---w stream-filter-ref
      |        |          stream-filter-ref
      |        +--:(within-subscription)
      |           +---w (filter-spec)?
      |              +--:(stream-subtree-filter)
      |              |  +---w stream-subtree-filter?   <anydata>
      |              |          {subtree}?
      |              +--:(stream-xpath-filter)
      |                 +---w stream-xpath-filter?
      |                         yang:xpath1.0 {xpath}?
      +---w stop-time?               yang:date-and-time
```

Figure 4: modify-subscription RPC tree diagram

If the publisher accepts the requested modifications on a currently suspended subscription, the subscription will immediately be resumed (i.e., the modified subscription is returned to the active state.)

The publisher MAY immediately suspend this newly modified subscription through the "subscription-suspended" notification before any event records are sent.

If the publisher rejects the RPC request, the subscription remains as prior to the request. That is, the request has no impact whatsoever. Rejection of the RPC for any reason is indicated by via RPC error as described in Section 2.4.6. The contents of such a rejected RPC MAY include hints on inputs which (if considered) may result in a successfully modified subscription. These hints MUST be transported within a yang-data "modify-subscription-stream-error-info" container inserted into the RPC error response.

Below is a tree diagram for "modify-subscription-RPC-yang-data". All objects contained in this tree are described within the included YANG model within Section 4.

```
yang-data modify-subscription-stream-error-info
   +--ro modify-subscription-stream-error-info
      +--ro reason?                identityref
      +--ro filter-failure-hint?   string
```

Figure 5: modify-subscription RPC yang-data tree diagram

2.4.4. Deleting a Dynamic Subscription

The "delete-subscription" operation permits canceling an existing subscription. If the publisher accepts the request, and the publisher has indicated success, the publisher MUST NOT send any more notification messages for this subscription. If the delete request matches a known subscription established on the same transport session, then it MUST be deleted; otherwise it MUST be rejected with no changes to the publisher.

Below is a tree diagram for "delete-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---x delete-subscription
   +---w input
      +---w identifier    subscription-id
```

Figure 6: delete-subscription RPC tree diagram

Dynamic subscriptions can only be deleted via this RPC using the same transport session previously used for subscription establishment. Configured subscriptions cannot be deleted using RPCs.

2.4.5. Killing a Dynamic Subscription

The "kill-subscription" operation permits an operator to end a dynamic subscription which is not associated with the transport session used for the RPC. A publisher MUST terminate any dynamic subscription identified by RPC request.

Configured subscriptions cannot be killed 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.

Below is a tree diagram for "kill-subscription". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---x kill-subscription
  +---w input
     +---w identifier    subscription-id
```

Figure 7: kill-subscription RPC tree diagram

2.4.6. RPC Failures

Whenever an RPC is unsuccessful, the publisher returns relevant information as part of the RPC error response. Transport level error processing MUST be done before RPC error processing described in this section. In all cases, RPC error information returned will use existing transport layer RPC structures, such as those seen with NETCONF in [RFC6241] Appendix A, or with RESTCONF in [RFC8040] Section 7.1. These structures MUST be able to encode subscription specific errors identified below and defined within this document’s YANG model.

As a result of this mixture, how subscription errors are encoded within an RPC error response is transport dependent. Following are valid errors which can occur for each RPC:

```
establish-subscription         modify-subscription
-----------------------        -------------------
dscp-unavailable               filter-unsupported
encoding-unsupported           insufficient-resources
filter-unsupported             no-such-subscription
history-unavailable
insufficient-resources
replay-unsupported

delete-subscription            kill-subscription
-----------------------        -----------------------
no-such-subscription            no-such-subscription
```

To see a NETCONF based example of an error response from above, see [I-D.draft-ietf-netconf-netconf-event-notifications], Figure 10.

There is one final set of transport independent RPC error elements included in the YANG model. These are the following three yang-data structures for failed event stream subscriptions:

1. "establish-subscription-stream-error-info": This MUST be returned if an RPC error reason has not been placed elsewhere within the transport portion of a failed "establish-subscription" RPC response. This MUST be sent if hints on how to overcome the RPC error are included.

2. "modify-subscription-stream-error-info": This MUST be returned if an RPC error reason has not been placed elsewhere within the transport portion of a failed "modify-subscription" RPC response. This MUST be sent if hints on how to overcome the RPC error are included.

3. "delete-subscription-error-info": This MUST be returned if an RPC error reason has not been placed elsewhere within the transport portion of a failed "delete-subscription" or "kill-subscription" RPC response.

2.5. Configured Subscriptions

A configured subscription is a subscription installed via configuration. Configured subscriptions may be modified by any configuration client with the proper permissions. Subscriptions can be modified or terminated via configuration at any point of their lifetime. Multiple configured subscriptions MUST be supportable over a single transport session.

Configured subscriptions have several characteristics distinguishing them from dynamic subscriptions:

- o persistence across publisher reboots,

- o persistence even when transport is unavailable, and

- o an ability to send notification messages to more than one receiver (note that receivers are unaware of the existence of any other receivers.)

On the publisher, supporting configured subscriptions is optional and advertised using the "configured" feature. On a receiver of a configured subscription, support for dynamic subscriptions is optional except where replaying missed event records is required.

In addition to the subscription parameters available to dynamic subscriptions described in Section 2.4.2, the following additional parameters are also available to configured subscriptions:

- o A "transport" which identifies the transport protocol to use to connect with all subscription receivers.

- o One or more receivers, each intended as the destination for event records. Note that each individual receiver is identifiable by its "name". This "name" plus the "transport" are used by a publisher implementation to a parameters needed to establish and maintain a network connection using that transport.

- o Optional parameters to identify where traffic should egress a publisher:

  - * A "source-interface" which identifies the egress interface to use from the publisher. Publisher support for this is optional and advertised using the "interface-designation" feature.

  - * A "source-address" address, which identifies the IP address to stamp on notification messages destined for the receiver.

  - * A "source-vrf" which identifies the VRF on which to reach receivers. This VRF is a network instance as defined within [I-D.draft-ietf-rtgwg-ni-model]. Publisher support for VRFs is optional and advertised using the "supports-vrf" feature.

  If none of the above parameters are set, notification messages MUST egress the publisher’s default interface.

A tree diagram describing these parameters is shown in Figure 20 within Section 3.3. All parameters are described within the YANG model in Section 4.

2.5.1. Configured Subscription State Model

Below is the state machine for a configured subscription on the publisher. This state machine describes the three states (valid, invalid, and concluded), as well as the transitions between these states. Start and end states are depicted to reflect configured subscription creation and deletion events. The creation or modification of a configured subscription initiates an evaluation by the publisher to determine if the subscription is in valid or invalid states. The publisher uses its own criteria in making this determination. If in the valid state, the subscription becomes operational. See (1) in the diagram below.

```
.........
: start :-.
:.......: |
     create  .---modify-----.----------------------------------.
          |  |              |                                  |
          V  V          .-------.         .......         .---------.
  .----[evaluate]--no--->|invalid|-delete->: end :<-delete-|concluded|
  |                      '-------'         :.....:         '---------'
  |-[evaluate]--no-(2).      ^                ^                ^
  |        ^          |      |                |                |
 yes       |          '->unsupportable      delete           stop-time
  |      modify         (subscription-   (subscription-   (subscription-
  |        |             terminated*)     terminated*)      concluded*)
  |        |                |                |                |
 (1)       |               (3)              (4)              (5)
  |   .-------------------------------------------------------------.
  '-->|                         valid                               |
      '-------------------------------------------------------------'

 Legend:
 dotted boxes: subscription added or removed via configuration
 dashed boxes: states for a subscription
 [evaluate]: decision point on whether the subscription is supportable
 (*): resulting subscription state change notification
```

Figure 8: Publisher state model for a configured subscription

A subscription in the valid state may move to the invalid state in one of two ways. First, it may be modified in a way which fails a re-evaluation. See (2) in the diagram. Second, the publisher might determine that the subscription is no longer supportable. This could be for reasons of an unexpected but sustained increase in an event stream's event records, degraded CPU capacity, a more complex referenced filter, or other higher priority subscriptions which have usurped resources. See (3) in the diagram. No matter the case, a

"subscription-terminated" notification is sent to any receivers in an active or suspended state. A subscription in the valid state may also transition to the concluded state via (5) if a configured stop time has been reached. In this case, a "subscription-concluded" notification is sent to any receivers in active or suspended states. Finally, a subscription may be deleted by configuration (4).

When a subscription is in the valid state, a publisher will attempt to connect with all receivers of a configured subscription and deliver notification messages. Below is the state machine for each receiver of a configured subscription. This receiver state machine is fully contained within the state machine of the configured subscription, and is only relevant when the configured subscription is in the valid state.

```
.-----------------------------------------------------------------.
|                          valid                                  |
|   .----------.                          .--------.              |
|   | receiver |---timeout---------------->|receiver|              |
|   |connecting|<----------------reset--(c)|timeout |              |
|   |          |<-transport                '--------'              |
|   '----------'  loss,reset-----------------------------.         |
|      (a)          |                                   |         |
|  subscription-   (b)                                 (b)        |
|  started*    .--------.                            .---------. |
|       '----->|        |(d)-insufficient CPU,------->|         | |
|              |receiver|    buffer overflow          |receiver | |
| subscription-| active |                             |suspended| |
|   modified*  |        |<----CPU, b/w sufficient,-(e)|         | |
|        '---->'--------'     subscription-modified*  '---------' |
'-----------------------------------------------------------------'
```

Legend:
dashed boxes which include the word 'receiver' show the possible states for an individual receiver of a valid configured subscription.
* indicates a state change notification

Figure 9: Receiver state for a configured subscription on a Publisher

When a configured subscription first moves to the valid state, the "state" leaf of each receiver is initialized to "connecting". If transport connectivity is not available to any receiver and there are any notification messages to deliver, a transport session is established (e.g., through [RFC8071]). Individual receivers are moved to the active state when a "subscription-started" state change notification is successfully passed to that receiver (a). Event records are only sent to active receivers. Receivers of a configured subscription remain active if both transport connectivity can be

verified to the receiver, and event records are not being dropped due to a publisher buffer overflow. The result is that a receiver will remain active on the publisher as long as events aren't being lost, or the receiver cannot be reached. In addition, a configured subscription's receiver MUST be moved to connecting if transport connectivity cannot be achieved, or if the receiver is reset via the "reset" action (b), (c). For more on reset, see Section 2.5.5.

A configured subscription's receiver MUST be moved to the suspended state if there is transport connectivity between the publisher and receiver, but notification messages are failing to be delivered due to publisher buffer overflow, or notification messages are not able to be generated for that receiver due to insufficient CPU (d). This is indicated to the receiver by the "subscription-suspended" state change notification.

A configured subscription receiver MUST be returned to the active state from the suspended state when notification messages are able to be generated, bandwidth is sufficient to handle the notification messages, and a receiver has successfully been sent a "subscription-resumed" or "subscription-modified" state change notification (e). The choice as to which of these two state change notifications is sent is determined by whether the subscription was modified during the period of suspension.

Modification of a configured subscription is possible at any time. A "subscription-modified" state change notification will be sent to all active receivers, immediately followed by notification messages conforming to the new parameters. Suspended receivers will also be informed of the modification. However this notification will await the end of the suspension for that receiver (e).

The mechanisms described above are mirrored in the RPCs and notifications within the document. It should be noted that these RPCs and notifications have been designed to be extensible and allow subscriptions into targets other than event streams. For instance, the YANG module defined in Section 5 of [I-D.ietf-netconf-yang-push] augments "/sn:modify-subscription/sn:input/sn:target".

2.5.2. Creating a Configured Subscription

Configured subscriptions are established using configuration operations against the top-level "subscriptions" subtree.

Because there is no explicit association with an existing transport session, configuration operations require additional parameters beyond those of dynamic subscriptions to indicate receivers, and

possibly whether the notification messages need to come from a specific egress interface on the publisher.

After a subscription is successfully established, the publisher immediately sends a "subscription-started" state change notification to each receiver. 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 specific call-home operations will be used to establish the connection. When transport connectivity is available, notification messages may then be pushed.

With active configured subscriptions, it is allowable to buffer event records even after a "subscription-started" has been sent. However if events are lost (rather than just delayed) due to replay buffer overflow, a new "subscription-started" must be sent. This new "subscription-started" indicates an event record discontinuity.

To see an example of subscription creation using configuration operations over NETCONF, see Appendix A of [I-D.draft-ietf-netconf-netconf-event-notifications].

Note that is possible to configure replay on a configured subscription. This capability is to allow a configured subscription to exist on a system so that event records generated during and following boot can be buffered and pushed as soon as the transport session is established.

2.5.3. Modifying a Configured Subscription

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

If the modification involves adding receivers, added receivers are placed in the connecting state. If a receiver is removed, the state change notification "subscription-terminated" is sent to that receiver if that receiver is active or suspended.

If the modification involves changing the policies for the subscription, the publisher sends to currently active receivers a "subscription-modified" notification. For any suspended receivers, a "subscription-modified" notification will be delayed until the receiver is resumed. (Note: in this case, the "subscription-modified" notification informs the receiver that the subscription has been resumed, so no additional "subscription-resumed" need be sent. Also note that if multiple modifications have occurred during the suspension, only the latest one need be sent to the receiver.)

2.5.4. Deleting a Configured Subscription

Subscriptions can be deleted through configuration against the top-level "subscriptions" subtree.

Immediately after a subscription is successfully deleted, the publisher sends to all receivers of that subscription a state change notification stating the subscription has ended (i.e., "subscription-terminated").

2.5.5. Resetting a Configured Subscription Receiver

It is possible that a configured subscription to a receiver needs to be reset. This is accomplished via the "reset" action within the YANG model at "/subscriptions/subscription/receivers/receiver/reset". This re-initialization may be useful in cases where a publisher has timed out trying to reach a receiver. When such a reset occurs, a transport session will be initiated if necessary, and a new "subscription-started" notification will be sent. This action does not have any effect on transport connectivity if the needed connectivity already exists.

2.5.6. Replay for a Configured Subscription

It is possible to do replay on a configured subscription. This is supported via the configuration of the "configured-replay" object on the subscription. The setting of this object enables the streaming of the buffered events for the subscribed stream. All buffered event which have been retained since the last publisher restart will be sent.

Replay of events records created since restart is useful. It allows event records generated before transport connectivity establishment to be passed to a receiver. Setting the restart time as the earliest configured replay time precludes possibility of resending of event records logged prior to publisher restart. It also ensures the same records will be sent to each configured receiver, regardless of the speed of transport connectivity establishment to each receiver. Finally, establishing restart as the earliest potential time for event records to be included within notification messages, a well-understood timeframe for replay is defined.

As a result, when any configured subscription receivers become active, buffered event records will be sent immediately after the "subscription-started" notification. The leading event record sent will be the first event record subsequent to the latest of three different times: the "replay-log-creation-time", "replay-log-aged-time", or the most recent publisher boot time. The "replay-log-

creation-time" and "replay-log-aged-time" are discussed in Section 2.4.2.1, and "replay-start-time" in Section 2.7.1. The most recent publisher boot time ensures that duplicate event records are not replayed from a previous time the publisher was booted.

It is quite possible that a receiver might want to retrieve event records from a stream prior to the latest boot. If such records exist where there is a configured replay, the publisher MUST send the time of the event record immediately preceding the "replay-start-time" within the "replay-previous-event-time" leaf. Through the existence of the "replay-previous-event-time", the receiver will know that earlier events prior to reboot exist. In addition, if the subscriber was previously receiving event records with the same subscription id, the receiver can determine if there was a timegap where records generated on the publisher were not successully received. And with this information, the receiver may choose to dynamically subscribe to retrieve any event records placed into the stream before the most recent boot time.

All other replay functionality remains the same as with dynamic subscriptions as described in Section 2.4.2.1.

2.6. Event Record Delivery

Whether dynamic or configured, once a subscription has been set up, the publisher streams event records via notification messages per the terms of the subscription. For dynamic subscriptions, notification messages are sent over the session used to establish the subscription. For configured subscriptions, notification messages are sent over the connections specified by the transport and each receiver of a configured subscription.

A notification message is sent to a receiver when an event record is not blocked by either the specified filter criteria or receiver permissions. This notification message MUST be encoded in a <notification> message as defined within [RFC5277], Section 4. And per [RFC5277]’s "eventTime" object definition, the "eventTime" is populated with the event occurrence time.

The following example within [RFC7950] section 7.16.3 is an example of a compliant message:

```
<notification
       xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2007-09-01T10:00:00Z</eventTime>
    <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 10: subscribed notification message

When a dynamic subscription has been started or modified, with "establish-subscription" or "modify-subscription" respectively, event records matching the newly applied filter criteria MUST NOT be sent until after the RPC reply has been sent.

When a configured subscription has been started or modified, event records matching the newly applied filter criteria MUST NOT be sent until after the "subscription-started" or "subscription-modified" notifications has been sent, respectively.

2.7. Subscription State Notifications

In addition to sending event records to receivers, a publisher MUST also send subscription state notifications when events related to subscription management have occurred.

Subscription state notifications are unlike other notifications in that they are never included in any stream. Instead, they are inserted (as defined in this section) within the sequence of notification messages sent to a particular receiver. Subscription state notifications cannot be filtered out, they cannot be stored in replay buffers, and they are delivered only to impacted receivers of a subscription. The identification of subscription state notifications is easy to separate from other notification messages through the use of the YANG extension "subscription-state-notif". This extension tags a notification as a subscription state notification.

The complete set of subscription state notifications is described in the following subsections.

2.7.1. subscription-started

This notification indicates that a configured subscription has started, and event records may be sent. Included in this state change notification are all the parameters of the subscription,

except for the receiver(s) addressing information and origin information indicating where notification messages will egress the publisher. Note that if a referenced filter from the "filters" container has been used within the subscription, the notification still provides the contents of that referenced filter under the "within-subscription" subtree.

Note that for dynamic subscriptions, no "subscription-started" notifications are ever sent.

Below is a tree diagram for "subscription-started". All objects contained in this tree are described within the included YANG model within Section 4.

```
      +---n subscription-started {configured}?
         +--ro identifier
         |       subscription-id
         +--ro (target)
         |  +--:(stream)
         |     +--ro (stream-filter)?
         |     |  +--:(by-reference)
         |     |  |  +--ro stream-filter-ref
         |     |  |          stream-filter-ref
         |     |  +--:(within-subscription)
         |     |     +--ro (filter-spec)?
         |     |        +--:(stream-subtree-filter)
         |     |        |  +--ro stream-subtree-filter?   <anydata>
         |     |        |          {subtree}?
         |     |        +--:(stream-xpath-filter)
         |     |           +--ro stream-xpath-filter?     yang:xpath1.0
         |     |                  {xpath}?
         |     +--ro stream                                stream-ref
         |     +--ro replay-start-time?
         |     |       yang:date-and-time {replay}?
         |     +--ro replay-previous-event-time?
         |             yang:date-and-time {replay}?
         +--ro stop-time?
         |       yang:date-and-time
         +--ro dscp?                                       inet:dscp
         |                                                 {dscp}?
         +--ro weighting?                                  uint8 {qos}?
         +--ro dependency?
         |       subscription-id {qos}?
         +--ro transport                                   transport
         |       {configured}?
         +--ro encoding?                                   encoding
         +--ro purpose?                                    string
                                                            {configured}?
```

Figure 11: subscription-started notification tree diagram

2.7.2. subscription-modified

This notification indicates that a subscription has been modified by configuration operations. It is delivered directly after the last event records processed using the previous subscription parameters, and before any event records processed after the modification.

Below is a tree diagram for "subscription-modified". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-modified
   +--ro identifier               subscription-id
   +--ro (target)
   |  +--:(stream)
   |     +--ro (stream-filter)?
   |     |  +--:(by-reference)
   |     |  |  +--ro stream-filter-ref        stream-filter-ref
   |     |  +--:(within-subscription)
   |     |     +--ro (filter-spec)?
   |     |        +--:(stream-subtree-filter)
   |     |        |  +--ro stream-subtree-filter?   <anydata>
   |     |        |          {subtree}?
   |     |        +--:(stream-xpath-filter)
   |     |           +--ro stream-xpath-filter?     yang:xpath1.0
   |     |                   {xpath}?
   |     +--ro stream                   stream-ref
   |     +--ro replay-start-time?       yang:date-and-time
   |             {replay}?
   +--ro stop-time?               yang:date-and-time
   +--ro dscp?                    inet:dscp {dscp}?
   +--ro weighting?               uint8 {qos}?
   +--ro dependency?              subscription-id {qos}?
   +--ro transport                transport {configured}?
   +--ro encoding?                encoding
   +--ro purpose?                 string {configured}?
```

Figure 12: subscription-modified notification tree diagram

A publisher most often sends this notification directly after the modification of any configuration parameters impacting a configured subscription. But it may also be sent at two other times:

1. Where a configured subscription has been modified during the suspension of a receiver, the notification will be delayed until the receiver’s suspension is lifted. In this situation, the notification indicates that the subscription has been both modified and resumed.

2. While this state change will most commonly be used with configured subscriptions, with dynamic subscriptions, there is also one time this notification will be sent. A "subscription-modified" state change notification MUST be sent if the contents of the filter identified by the subscription’s "stream-filter-ref" leaf has changed.

2.7.3. subscription-terminated

This notification indicates that no further event records for this subscription should be expected from the publisher. A publisher may terminate the sending event records to a receiver for the following reasons:

1. Configuration which removes a configured subscription, or a "kill-subscription" RPC which ends a dynamic subscription. These are identified via the reason "no-such-subscription".

2. A referenced filter is no longer accessible. This is identified by "filter-unavailable".

3. The event stream referenced by a subscription is no longer accessible by the receiver. This is identified by "stream-unavailable".

4. A suspended subscription has exceeded some timeout. This is identified by "suspension-timeout".

Each of the reasons above correspond one-to-one with a "reason" identityref specified within the YANG model.

Below is a tree diagram for "subscription-terminated". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-terminated
   +--ro identifier    subscription-id
   +--ro reason        identityref
```

Figure 13: subscription-terminated notification tree diagram

Note: this state change notification MUST be sent to a dynamic subscription’s receiver when the subscription ends unexpectedly. The cases when this might happen are when a "kill-subscription" RPC is successful, or when some other event not including the reaching the subscription’s "stop-time" results in a publisher choosing to end the subscription.

2.7.4. subscription-suspended

This notification indicates that a publisher has suspended the sending of event records to a receiver, and also indicates the possible loss of events. Suspension happens when capacity

constraints stop a publisher from serving a valid subscription. The two conditions where is this possible are:

1. "insufficient-resources" when a publisher is unable to produce the requested event stream of notification messages, and

2. "unsupportable-volume" when the bandwidth needed to get generated notification messages to a receiver exceeds a threshold.

These conditions are encoded within the "reason" object. No further notification will be sent until the subscription resumes or is terminated.

Below is a tree diagram for "subscription-suspended". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-suspended
   +--ro identifier    subscription-id
   +--ro reason        identityref
```

Figure 14: subscription-suspended notification tree diagram

2.7.5. subscription-resumed

This notification indicates that a previously suspended subscription has been resumed under the unmodified terms previously in place. Subscribed event records generated after the issuance of this state change notification may now be sent.

Below is the tree diagram for "subscription-resumed". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-resumed
   +--ro identifier    subscription-id
```

Figure 15: subscription-resumed notification tree diagram

2.7.6. subscription-completed

This notification indicates that a subscription that includes a "stop-time" has successfully finished passing event records upon the reaching of that time.

Below is a tree diagram for "subscription-completed". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n subscription-completed
   +--ro identifier    subscription-id
```

Figure 16: subscription-completed notification tree diagram

2.7.7. replay-completed

This notification indicates that all of the event records prior to the current time have been passed to a receiver. It is sent before any notification message containing an event record with a timestamp later than (1) the "stop-time" or (2) the subscription’s start time.

If a subscription contains no "stop-time", or has a "stop-time" that has not been reached, then after the "replay-completed" notification has been sent, additional event records will be sent in sequence as they arise naturally on the publisher.

Below is a tree diagram for "replay-completed". All objects contained in this tree are described within the included YANG model within Section 4.

```
+---n replay-completed
   +--ro identifier    subscription-id
```

Figure 17: replay-completed notification tree diagram

2.8. Subscription Monitoring

In the operational datastore, the container "subscriptions" maintains the state of all dynamic subscriptions, as well as all configured subscriptions. Using datastore retrieval operations, or subscribing to the "subscriptions" container [I-D.ietf-netconf-yang-push] allows the state of subscriptions and their connectivity to receivers to be monitored.

Each subscription in the operational datastore is represented as a list element. Included in this list are event counters for each receiver, the state of each receiver, as well as the subscription parameters currently in effect. The appearance of the leaf "configured-subscription-state" indicates that a particular subscription came into being via configuration. This leaf also indicates if current state of that subscription is valid, invalid, and concluded.

To understand the flow of event records within a subscription, there are two counters available for each configured and dynamic receiver. The first counter is "count-sent" which shows the quantity of events actually identified for sending to a receiver. The second counter is "count-excluded" which shows event records not sent to receiver. "count-excluded" shows the combined results of both access control and per-subscription filtering. For configured subscriptions, counters are reset whenever the subscription is evaluated to valid (see (1) in Figure 8).

Dynamic subscriptions are removed from the operational datastore once they expire (reaching stop-time) or when they are terminated. While many subscription objects are shown as configurable, dynamic subscriptions are only included within the operational datastore and as a result are not configurable.

2.9. Advertisement

Publishers supporting this document MUST indicate support of the YANG model "ietf-subscribed-notifications" within the YANG library of the publisher. In addition support for optional features "encode-xml", "encode-json", "configured" "supports-vrf", "qos", "xpath", "subtree", "interface-designation", "dscp", and "replay" MUST be indicated if supported.

3. YANG Data Model Trees

This section contains tree diagrams for nodes defined in Section 4. For tree diagrams of state change notifications, see Section 2.7. Or for the tree diagrams for the RPCs, see Section 2.4.

3.1. Event Streams Container

A publisher maintains a list of available event streams as operational data. This list contains both standardized and vendor-specific event streams. This enables subscribers to discover what streams a publisher supports.

```
+--ro streams
   +--ro stream* [name]
      +--ro name                       string
      +--ro description                string
      +--ro replay-support?            empty {replay}?
      +--ro replay-log-creation-time   yang:date-and-time {replay}?
      +--ro replay-log-aged-time?      yang:date-and-time {replay}?
```

Figure 18: Stream Container tree diagram

Above is a tree diagram for the streams container. All objects contained in this tree are described within the included YANG model within Section 4.

3.2. Filters Container

The "filters" container maintains a list of all subscription filters that persist outside the life-cycle of a single subscription. This enables pre-defined filters which may be referenced by more than one subscription.

```
+--rw filters
   +--rw stream-filter* [identifier]
      +--rw identifier               filter-id
      +--rw (filter-spec)?
         +--:(stream-subtree-filter)
         |  +--rw stream-subtree-filter?   <anydata> {subtree}?
         +--:(stream-xpath-filter)
            +--rw stream-xpath-filter?     yang:xpath1.0 {xpath}?
```

Figure 19: Filter Container tree diagram

Above is a tree diagram for the filters container. All objects contained in this tree are described within the included YANG model within Section 4.

3.3. Subscriptions Container

The "subscriptions" container maintains 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.

```
 +--rw subscriptions
    +--rw subscription* [identifier]
       +--rw identifier
       |       subscription-id
       +--rw (target)
       |  +--:(stream)
       |     +--rw (stream-filter)?
       |     |  +--:(by-reference)
       |     |  |  +--rw stream-filter-ref
       |     |  |          stream-filter-ref
       |     |  +--:(within-subscription)
       |     |     +--rw (filter-spec)?
       |     |        +--:(stream-subtree-filter)
       |     |        |  +--rw stream-subtree-filter?   <anydata>
       |     |        |          {subtree}?
```

```
          |     |        +--:(stream-xpath-filter)
          |     |           +--rw stream-xpath-filter?     yang:xpath1.0
          |     |                   {xpath}?
          |     +--rw stream                                stream-ref
          |     +--ro replay-start-time?
          |     |       yang:date-and-time {replay}?
          |     +--rw configured-replay?                    empty
          |             {configured,replay}?
          +--rw stop-time?
          |       yang:date-and-time
          +--rw dscp?                                       inet:dscp
          |       {dscp}?
          +--rw weighting?                                  uint8 {qos}?
          +--rw dependency?
          |       subscription-id {qos}?
          +--rw transport                                   transport
          |       {configured}?
          +--rw encoding?                                   encoding
          +--rw purpose?                                    string
          |       {configured}?
          +--rw (notification-message-origin)? {configured}?
          |  +--:(interface-originated)
          |  |  +--rw source-interface?
          |  |          if:interface-ref {interface-designation}?
          |  +--:(address-originated)
          |     +--rw source-vrf?
          |     |       -> /ni:network-instances/network-instance/name
          |     |       {supports-vrf}?
          |     +--rw source-address?
          |             inet:ip-address-no-zone
          +--ro configured-subscription-state?              enumeration
          |       {configured}?
          +--rw receivers
             +--rw receiver* [name]
                +--rw name              string
                +--ro count-sent?       yang:zero-based-counter64
                +--ro count-excluded?   yang:zero-based-counter64
                +--ro state             enumeration
                +---x reset {configured}?
                   +--ro output
                      +--ro time    yang:date-and-time
```

Figure 20: Subscriptions tree diagram

Above is a tree diagram for the subscriptions container. All objects contained in this tree are described within the included YANG model within Section 4.

4. Data Model

This module imports typedefs from [RFC6991], [RFC8343], and [RFC8040], and it references [I-D.draft-ietf-rtgwg-ni-model], [XPATH], [RFC6241], [RFC7540], [RFC7951] and [RFC7950].

[note to the RFC Editor - please replace XXXX within this YANG model with the number of this document, and XXXY with the number of [I-D.draft-ietf-rtgwg-ni-model]]

[note to the RFC Editor - please replace the two dates within the YANG module with the date of publication]

```
<CODE BEGINS> file "ietf-subscribed-notifications@2018-07-02.yang"
module ietf-subscribed-notifications {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications";

  prefix sn;

  import ietf-inet-types {
     prefix inet;
     reference
       "RFC 6991: Common YANG Data Types";
  }
  import ietf-interfaces {
    prefix if;
    reference
       "RFC 8343: A YANG Data Model for Interface Management";
  }
  import ietf-network-instance {
    prefix ni;
    reference
       "draft-ietf-rtgwg-ni-model-12: YANG Model for Network Instances";
  }
  import ietf-restconf   {
    prefix rc;
    reference
       "RFC 8040: RESTCONF Protocol";
  }
  import ietf-yang-types {
    prefix yang;
    reference
       "RFC 6991: Common YANG Data Types";
  }

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

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

     Author:   Alexander Clemm
               <mailto:ludwig@clemm.org>

     Author:   Eric Voit
               <mailto:evoit@cisco.com>

     Author:   Alberto Gonzalez Prieto
               <mailto:agonzalezpri@vmware.com>

     Author:   Einar Nilsen-Nygaard
               <mailto:einarnn@cisco.com>

     Author:   Ambika Prasad Tripathy
               <mailto:ambtripa@cisco.com>";

  description
    "Contains a YANG specification for subscribing to event records
    and receiving matching content within notification messages.

    Copyright (c) 2018 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
    (https://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 2018-07-02 {
    description
      "Initial version";
    reference
    "RFC XXXX: Customized Subscriptions to a Publisher's Event Streams";
  }

  /*
   * FEATURES
   */

  feature configured {
    description
```

```
      "This feature indicates that configuration of subscription is
      supported.";
  }

  feature dscp {
    description
      "This feature indicates a publisher supports the placement of
      suggested prioritization levels for network transport within
      notification messages.";
  }

  feature encode-json {
    description
      "This feature indicates that JSON encoding of notification
       messages is supported.";
  }

  feature encode-xml {
    description
      "This feature indicates that XML encoding of notification
       messages is supported.";
  }

  feature interface-designation {
    description
      "This feature indicates a publisher supports sourcing all receiver
      interactions for a configured subscription from a single
      designated egress interface.";
  }

  feature qos {
    description
      "This feature indicates a publisher supports absolute dependencies
      of one subscription’s traffic over another, as well as weighted
      bandwidth sharing between subscriptions.  Both of these are
      Quality of Service (QoS) features which allow differentiated
      treatment of notification messages between a publisher and a
      specific receiver.";
  }

  feature replay {
    description
      "This feature indicates that historical event record replay is
      supported.  With replay, it is possible for past event records to
      be streamed in chronological order.";
  }

  feature subtree {
```

```
    description
      "This feature indicates support for YANG subtree filtering.";
    reference "RFC 6241, Section 6.";
  }

  feature supports-vrf {
    description
      "This feature indicates a publisher supports VRF configuration
      for configured subscriptions.  VRF support for dynamic
      subscriptions does not require this feature.";
    reference "RFC XXXY, Section 6.";
  }

  feature xpath {
    description
      "This feature indicates support for xpath filtering.";
    reference "http://www.w3.org/TR/1999/REC-xpath-19991116";
  }

  /*
   * EXTENSIONS
   */

  extension subscription-state-notification {
    description
      "This statement applies only to notifications. It indicates that
       the notification is a subscription state notification. Therefore
       it does not participate in a regular event stream and does not
       need to be specifically subscribed to in order to be received.
       This statement can only occur as a substatement to the YANG
       'notification' statement.  This statement is not for use outside
       of this YANG module.";
  }

  /*
   * IDENTITIES
   */

  /* Identities for RPC and Notification errors */

  identity delete-subscription-error {
     description
      "Problem found while attempting to fulfill either a
      'delete-subscription' RPC request or a 'kill-subscription'
      RPC request.";
  }

  identity establish-subscription-error {
```

```
     description
      "Problem found while attempting to fulfill an
      'establish-subscription' RPC request.";
  }

  identity modify-subscription-error {
     description
      "Problem found while attempting to fulfill a
      'modify-subscription' RPC request.";
  }

  identity subscription-suspended-reason {
     description
      "Problem condition communicated to a receiver as part of a
      'subscription-terminated' notification.";
  }

  identity subscription-terminated-reason {
     description
      "Problem condition communicated to a receiver as part of a
      'subscription-terminated' notification.";
  }

  identity dscp-unavailable {
    base establish-subscription-error;
    if-feature "dscp";
    description
      "The publisher is unable mark notification messages with a
      prioritization information in a way which will be respected during
      network transit.";
  }

  identity encoding-unsupported {
    base establish-subscription-error;
    description
      "Unable to encode notification messages in the desired format.";
  }

  identity filter-unavailable {
    base subscription-terminated-reason;
    description
     "Referenced filter does not exist.  This means a receiver is
     referencing a filter which doesn't exist, or to which they do not
     have access permissions.";
  }

  identity filter-unsupported {
    base establish-subscription-error;
```

```
    base modify-subscription-error;
    description
     "Cannot parse syntax within the filter.  This failure can be from
     a syntax error, or a syntax too complex to be processed by the
     publisher.";
  }

  identity history-unavailable {
    base establish-subscription-error;
    if-feature "replay";
    description
     "Replay request too far into the past. This means the publisher
      does store historic information for the requested stream, but
      not back to the requested timestamp.";
  }

  identity insufficient-resources {
    base establish-subscription-error;
    base modify-subscription-error;
    base subscription-suspended-reason;
    description
      "The publisher has insufficient resources to support the
       requested subscription.  An example might be that allocated CPU
       is too limited to generate the desired set of notification
       messages.";
  }

  identity no-such-subscription {
    base modify-subscription-error;
    base delete-subscription-error;
    base subscription-terminated-reason;
    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 configured subscription.";
  }

  identity replay-unsupported {
    base establish-subscription-error;
    if-feature "replay";
    description
     "Replay cannot be performed for this subscription. This means the
      publisher will not provide the requested historic information from
      the event stream via replay to this receiver.";
  }

  identity stream-unavailable {
    base subscription-terminated-reason;
```

```
    description
     "Not a subscribable stream.  This means the referenced event stream
      is not available for subscription by the receiver.";
  }

  identity suspension-timeout {
    base subscription-terminated-reason;
    description
     "Termination of previously suspended subscription. The publisher
      has eliminated the subscription as it exceeded a time limit for
      suspension.";
  }

  identity unsupportable-volume {
    base subscription-suspended-reason;
    description
      "The publisher does not have the network bandwidth needed to get
      the volume of generated information intended for a receiver.";
  }

  /* Identities for encodings */

  identity configurable-encoding {
    description
      "If a transport identity derives from this identity, it means
       that it supports configurable encodings.";
  }

  identity encoding {
    description
      "Base identity to represent data encodings";
  }

  identity encode-xml {
    base encoding;
    if-feature "encode-xml";
    description
      "Encode data using XML as described in RFC 7950";
    reference
      "RFC 7950 - The YANG 1.1 Data Modeling Language";
  }

  identity encode-json {
    base encoding;
    if-feature "encode-json";
    description
      "Encode data using JSON as described in RFC 7951";
    reference
```

```
      "RFC 7951 - JSON Encoding of Data Modeled with YANG";
  }

  /* Identities for transports */
  identity transport {
    description
      "An identity that represents the underlying mechanism for
      passing notification messages.";
  }

  identity inline-address {
    description
      "A transport identity can derive from this identity in order to
       allow inline definition of the host address in the
       'receiver' list";
  }

  /*
   * TYPEDEFs
   */

  typedef encoding {
    type identityref {
      base encoding;
    }
    description
      "Specifies a data encoding, e.g. for a data subscription.";
  }

  typedef filter-id {
    type string;
    description
      "A type to identify filters which can be associated with a
       subscription.";
  }

  typedef stream-filter-ref {
    type leafref {
      path "/sn:filters/sn:stream-filter/sn:identifier";
    }
    description
      "This type is used to reference an event stream filter.";
  }

  typedef stream-ref {
    type leafref {
      path "/sn:streams/sn:stream/sn:name";
    }
```

```
    description
      "This type is used to reference a system-provided stream.";
  }

  typedef subscription-id {
    type uint32;
    description
      "A type for subscription identifiers.";
  }

  typedef transport {
    type identityref {
      base transport;
    }
    description
      "Specifies transport used to send notification messages to a
       receiver.";
  }

  /*
   * GROUPINGS
   */

  grouping stream-filter-elements {
    description
      "This grouping defines the base for filters applied to event
       streams.";
    choice filter-spec {
      description
        "The content filter specification for this request.";
      anydata stream-subtree-filter {
        if-feature "subtree";
        description
          "Event stream evaluation criteria encoded in the syntax of a
          subtree filter as defined in RFC 6241, Section 6.

          The subtree filter is applied to the representation of
          individual, delineated event records as contained within the
          event stream.  For example, if the notification message
          contains an instance of a notification defined in YANG, then
          the top-level element is the name of the YANG notification.

          If the subtree filter returns a non-empty node set, the filter
          matches the event record, and the event record is included in
          the notification message sent to the receivers.";
        reference "RFC 6241, Section 6.";
      }
      leaf stream-xpath-filter {
```

```
        if-feature "xpath";
        type yang:xpath1.0;
        description
          "Event stream evaluation criteria encoded in the syntax of
           an XPath 1.0 expression.

           The XPath expression is evaluated on the representation of
           individual, delineated event records as contained within
           the event stream.  For example, if the notification message
           contains an instance of a notification defined in YANG,
           then the top-level element is the name of the YANG
           notification, and the root node has this top-level element
           as the only child.

           The result of the XPath expression is converted to a
           boolean value using the standard XPath 1.0 rules.  If the
           boolean value is 'true', the filter matches the event record,
           and the event record is included in the notification message
           sent to the receivers.

           The expression is evaluated in the following XPath context:

             o  The set of namespace declarations are those in scope on
                the 'stream-xpath-filter' leaf element.

             o  The set of variable bindings is empty.

             o  The function library is the core function library, and
                the XPath functions defined in section 10 in RFC 7950.

             o  The context node is the root node.";
        reference
          "http://www.w3.org/TR/1999/REC-xpath-19991116
           RFC 7950, Section 10.";

      }
    }
  }

  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 {
      if-feature "dscp";
      type inet:dscp;
      default "0";
```

```
      description
        "The desired network transport priority level. This is the
         priority set on notification messages encapsulating the results
         of the subscription.  This transport priority is shared for all
         receivers of a given subscription.";
    }
    leaf weighting {
      if-feature "qos";
      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 {
      if-feature "qos";
      type subscription-id;
      description
        "Provides the 'subscription-id' of a parent subscription which
         has absolute precedence 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
         the parent has something ready to push.

         If a dependency is asserted via configuration or via RPC, but
         the referenced 'subscription-id' does not exist, the dependency
         is silently discarded.  If a referenced subscription is deleted
         this dependency is removed.";
      reference
        "RFC-7540, section 5.3.1";
    }
  }

  grouping subscription-policy-modifiable {
    description
      "This grouping describes all objects which may be changed
      in a subscription.";
    choice target {
      mandatory true;
      description
        "Identifies the source of information against which a
        subscription is being applied, as well as specifics on the
        subset of information desired from that source.";
      case stream {
```

```
        choice stream-filter {
          description
            "An event stream filter can be applied to a subscription.
            That filter will come either referenced from a global list,
            or be provided within the subscription itself.";
          case by-reference {
            description
              "Apply a filter that has been configured separately.";
            leaf stream-filter-ref {
              type stream-filter-ref;
              mandatory true;
              description
                "References an existing stream filter which is to
                be applied to an event stream for the subscription.";
            }
          }
          case within-subscription {
            description
              "Local definition allows a filter to have the same
              lifecycle as the subscription.";
            uses stream-filter-elements;
          }
        }
      }
    }
    leaf stop-time {
      type yang:date-and-time;
      description
        "Identifies a time after which notification messages for a
        subscription should not be sent.  If 'stop-time' is not present,
        the notification messages will continue until the subscription
        is terminated.  If 'replay-start-time' exists, 'stop-time' must
        be for a subsequent time. If 'replay-start-time' doesn't exist,
        'stop-time' when established must be for a future time.";
    }
  }

  grouping subscription-policy-dynamic {
    description
      "This grouping describes the only information concerning a
       subscription which can be passed over the RPCs defined in this
       model.";
    uses subscription-policy-modifiable {
      augment target/stream {
        description
          "Adds additional objects which can be modified by RPC.";
        leaf stream {
          type stream-ref {
```

```
            require-instance false;
          }
          mandatory true;
          description
            "Indicates the event stream to be considered for
            this subscription.";
        }
        leaf replay-start-time {
          if-feature "replay";
          type yang:date-and-time;
          config false;
          description
            "Used to trigger the replay feature for a dynamic
            subscription, with event records being selected needing to
            be at or after the start at the time specified.  If
            'replay-start-time' is not present, this is not a replay
            subscription and event record push should start immediately.
            It is never valid to specify start times that are later than
            or equal to the current time.";
        }
      }
    }
    uses update-qos;
  }

  grouping subscription-policy {
    description
      "This grouping describes the full set of policy information
      concerning both dynamic and configured subscriptions, with the
      exclusion of both receivers and networking information specific to
      the publisher such as what interface should be used to transmit
      notification messages.";
    uses subscription-policy-dynamic;
    leaf transport {
      if-feature "configured";
      type transport;
      mandatory true;
      description
        "This leaf specifies the transport used to deliver
        messages destined to all receivers of a subscription.";
    }
    leaf encoding {
      when 'not(../transport) or derived-from(../transport,
      "sn:configurable-encoding")';
      type encoding;
      description
        "The type of encoding for notification messages.   For a
        dynamic subscription, if not included as part of an establish-
```

```
        subscription RPC, the encoding will be populated with the
        encoding used by that RPC.  For a configured subscription, if
        not explicitly configured the encoding with be the default
        encoding for an underlying transport.";
    }
    leaf purpose {
      if-feature "configured";
      type string;
      description
        "Open text allowing a configuring entity to embed the
        originator or other specifics of this subscription.";
    }
  }

  /*
   * 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 occurs, or the
       publisher cannot meet the terms of a subscription, an RPC error
       is returned, the subscription is not created.  In that case, the
       RPC reply's 'error-info' MAY include suggested parameter settings
       that would have a higher likelihood of succeeding in a subsequent
       'establish-subscription' request.";
    input {
      uses subscription-policy-dynamic;
      leaf encoding {
        type encoding;
        description
          "The type of encoding for the subscribed data. If not
          included as part of the RPC, the encoding MUST be set by the
          publisher to be the encoding used by this RPC.";
      }
    }
    output {
      leaf identifier {
        type subscription-id;
        mandatory true;
        description
          "Identifier used for this subscription.";
      }
      leaf replay-start-time-revision {
```

```
        if-feature "replay";
        type yang:date-and-time;
          description
            "If a replay has been requested, this represents the
            earliest time covered by the event buffer for the requested
            stream.  The value of this object is the
            'replay-log-aged-time' if it exists.  Otherwise it is the
            'replay-log-creation-time'.  All buffered event records
            after this time will be replayed to a receiver.  This
            object will only be sent if the starting time has been
            revised to be later than the time requested by the
            subscriber.";
      }
    }
  }

  rc:yang-data establish-subscription-stream-error-info {
    container establish-subscription-stream-error-info {
      description
        "If any 'establish-subscription' RPC parameters are
        unsupportable against the event stream, a subscription is not
        created and the RPC error response MUST indicate the reason
        why the subscription failed to be created. This yang-data MAY be
        inserted as structured data within a subscription's RPC error
        response to indicate the failure reason.  This yang-data MUST be
        inserted if hints are to be provided back to the subscriber.";
      leaf reason {
        type identityref {
          base establish-subscription-error;
        }
        description
          "Indicates the reason why the subscription has failed to
          be created to a targeted stream.";
        }
      leaf filter-failure-hint {
        type string;
          description
            "Information describing where and/or why a provided filter
             was unsupportable for a subscription.";
      }
    }
  }

  rpc modify-subscription {
    description
      "This RPC allows a subscriber to modify a dynamic subscription's
       parameters.  If successful, the changed subscription
       parameters remain in effect for the duration of the subscription,
```

```
       until the subscription is again modified, or until the
       subscription is terminated.  In case of an error or an inability
       to meet the modified parameters, the subscription is not modified
       and the original subscription parameters remain in effect.
       In that case, the RPC error MAY include 'error-info' suggested
       parameter hints that would have a high likelihood of succeeding
       in a subsequent 'modify-subscription' request.  A successful
       'modify-subscription' will return a suspended subscription to an
       'active' state.";
    input {
      leaf identifier {
        type subscription-id;
        mandatory true;
        description
          "Identifier to use for this subscription.";
      }
      uses subscription-policy-modifiable;
    }
  }

  rc:yang-data modify-subscription-stream-error-info {
    container modify-subscription-stream-error-info {
      description
        "This yang-data MAY be provided as part of a subscription's RPC
        error response when there is a failure of a
        'modify-subscription' RPC which has been made against a
        stream.  This yang-data MUST be used if hints are to be
        provided back to the subscriber.";
      leaf reason {
        type identityref {
          base modify-subscription-error;
        }
        description
          "Information in a 'modify-subscription' RPC error response
          which indicates the reason why the subscription to an event
          stream has failed to be modified.";
      }
      leaf filter-failure-hint {
        type string;
          description
            "Information describing where and/or why a provided filter
             was unsupportable for a subscription.";
      }
    }
  }

  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.

       If an error occurs, the server replies with an 'rpc-error' where
       the 'error-info' field MAY contain an
       'delete-subscription-error-info' structure.";
    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' from the same origin as this RPC
           can be deleted via this RPC.";
      }
    }
  }

  rpc kill-subscription {
    description
      "This RPC allows an operator to delete a dynamic subscription
       without restrictions on the originating subscriber or underlying
       transport session.

       If an error occurs, the server replies with an 'rpc-error' where
       the 'error-info' field MAY contain an
       'delete-subscription-error-info' structure.";
    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.";
      }
    }
  }

  rc:yang-data delete-subscription-error-info {
    container delete-subscription-error-info {
      description
        "If a 'delete-subscription' RPC or a 'kill-subscription' RPC
        fails, the subscription is not deleted and the RPC error
        response MUST indicate the reason for this failure. This
        yang-data MAY be inserted as structured data within a
```

```
        subscription's RPC error response to indicate the failure
        reason.";
      leaf reason {
        type identityref {
          base delete-subscription-error;
        }
        mandatory true;
        description
          "Indicates the reason why the subscription has failed to be
          deleted.";
      }
    }
  }

  /*
   * NOTIFICATIONS
   */

  notification replay-completed {
    sn:subscription-state-notification;
    if-feature "replay";
    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 subscription-completed {
    sn:subscription-state-notification;
    if-feature "configured";
    description
      "This notification is sent to indicate that a subscription has
       finished passing event records, as the 'stop-time' has been
       reached.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the gracefully completed subscription.";
    }
  }
```

```
  notification subscription-modified {
    sn:subscription-state-notification;
    description
      "This notification indicates that a subscription has been
       modified.  Notification messages sent from this point on will
       conform to the modified terms of the subscription.  For
       completeness, this state change 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 {
      refine "target/stream/stream-filter/within-subscription" {
        description
          "Filter applied to the subscription.  If the
          'stream-filter-ref' is populated, the filter within the
          subscription came from the 'filters' container.  Otherwise it
          is populated in-line as part of the subscription.";
      }
    }
  }

  notification subscription-resumed {
    sn:subscription-state-notification;
    description
      "This notification indicates that a subscription that had
       previously been suspended has resumed. Notifications will once
       again be sent.  In addition, a 'subscription-resumed' indicates
       that no modification of parameters has occurred since the last
       time event records have been sent.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
  }

  notification subscription-started {
    sn:subscription-state-notification;
    if-feature "configured";
    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 {
      refine "target/stream/replay-start-time" {
         description
           "Indicates the time that a replay using for the streaming of
           buffered event records.  This will be populated with the most
           recent of the following: 'replay-log-creation-time',
           'replay-log-aged-time', 'replay-start-time', or the most
           recent publisher boot time.";
      }
      refine "target/stream/stream-filter/within-subscription" {
        description
          "Filter applied to the subscription.  If the
          'stream-filter-ref' is populated, the filter within the
          subscription came from the 'filters' container.  Otherwise it
          is populated in-line as part of the subscription.";
      }
      augment "target/stream" {
        description
          "This augmentation adds additional parameters specific to a
          subscription-started notification.";
        leaf replay-previous-event-time {
          when "../replay-start-time";
          if-feature "replay";
          type yang:date-and-time;
            description
            "If there is at least one event in the replay buffer prior
            to 'replay-start-time', this gives the time of the event
            generated immediately prior to the 'replay-start-time'.

            If a receiver previously received event records for this
            configured subscription, it can compare this time to the
            last event record previously received.  If the two are not
            the same (perhaps due to a reboot), then a dynamic replay
            can be initiated to acquire any missing event records.";
        }
      }
    }
  }

  notification subscription-suspended {
    sn:subscription-state-notification;
```

```
    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.";
    }
    leaf reason {
      type identityref {
        base subscription-suspended-reason;
      }
      mandatory true;
      description
        "Identifies the condition which resulted in the suspension.";
    }
  }

  notification subscription-terminated {
    sn:subscription-state-notification;
    description
      "This notification indicates that a subscription has been
       terminated.";
    leaf identifier {
      type subscription-id;
      mandatory true;
      description
        "This references the affected subscription.";
    }
    leaf reason {
      type identityref {
        base subscription-terminated-reason;
      }
      mandatory true;
      description
        "Identifies the condition which resulted in the termination .";
    }
  }


  /*
   * DATA NODES
   */
```

```
  container streams {
    config false;
    description
      "This container contains information on the built-in streams
      provided by the publisher.";
    list stream {
      key "name";
      description
        "Identifies the built-in event streams that are supported by the
         publisher.";
      leaf name {
        type string;
        description
          "A handle for a system-provided event stream made up of a
          sequential set of event records, each of which is
          characterized by its own domain and semantics.";
      }
      leaf description {
        type string;
        mandatory true;
        description
          "A description of the event stream, including such information
           as the type of event records that are available within this
           event stream.";
      }
      leaf replay-support {
        if-feature "replay";
        type empty;
        description
          "Indicates that event record replay is available on this
          stream.";
      }
      leaf replay-log-creation-time {
        when "../replay-support";
        if-feature "replay";
        type yang:date-and-time;
        mandatory true;
        description
          "The timestamp of the creation of the log used to support the
          replay function on this stream. This time might be earlier
          than the earliest available information contained in the log.
          This object is updated if the log resets for some reason.";
      }
      leaf replay-log-aged-time {
        if-feature "replay";
        type yang:date-and-time;
        description
          "The timestamp associated with last event record which has
```

```
            been aged out of the log.  This timestamp identifies how far
            back into history this replay log extends, if it doesn't
            extend back to the 'replay-log-creation-time'.  This object
            MUST be present if replay is supported and any event records
            have been aged out of the log.";
        }
      }
    }

    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 stream-filter {
        key "identifier";
        description
          "A list of pre-configured filters that can be applied to
          subscriptions.";
        leaf identifier {
          type filter-id;
          description
            "An identifier to differentiate between filters.";
        }
        uses stream-filter-elements;
      }
    }

    container subscriptions {
      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";
        description
          "The identity and specific parameters of a subscription.
           Subscriptions within this list can be created using a control
           channel or RPC, or be established through configuration.

           If configuration operations or the 'kill-subscription' RPC are
           used to delete a subscription, a 'subscription-terminated'
           message is sent to any active or suspended receivers.";
        leaf identifier {
          type subscription-id;
          description
```

```
          "Identifier of a subscription; unique within a publisher";
      }
      uses subscription-policy {
        refine "target/stream/stream" {
          description
            "Indicates the event stream to be considered for this
            subscription.  If an event stream has been removed,
            and no longer can be referenced by an active subscription,
            send a 'subscription-terminated' notification with
            'stream-unavailable' as the reason.  If a configured
            subscription refers to a non-existent stream, move that
            subscription to the 'invalid' state.";
        }
        augment "target/stream" {
          description
            "Enables objects to added to a configured stream
            subscription";
          leaf configured-replay {
            if-feature "configured";
            if-feature "replay";
            type empty;
            description
              "The presence of this leaf indicates that replay for the
              configured subscription should start at the earliest time
              in the event log, or at the publisher boot time, which
              ever is later.";
          }
        }
      }
      choice notification-message-origin {
        if-feature "configured";
        description
          "Identifies the egress interface on the publisher from which
           notification messages are to be sent.";
        case interface-originated {
          description
            "When notification messages to egress a specific, designated
             interface on the publisher.";
          leaf source-interface {
            if-feature "interface-designation";
            type if:interface-ref;
            description
              "References the interface for notification messages.";
          }
        }
        case address-originated {
          description
            "When notification messages are to depart from a publisher
```

```
             using specific originating address and/or routing context
             information.";
          leaf source-vrf {
            if-feature "supports-vrf";
            type leafref {
              path "/ni:network-instances/ni:network-instance/ni:name";
            }
            description
              "VRF from which notification messages should egress a
              publisher.";
          }
          leaf source-address {
            type inet:ip-address-no-zone;
            description
              "The source address for the notification messages.  If a
              source VRF exists, but this object doesn't, a publisher's
              default address for that VRF must be used.";
          }
        }
      }
      leaf configured-subscription-state {
        if-feature "configured";
        type enumeration {
          enum valid {
            value 1;
            description
              "Connection is active and healthy.";
          }
          enum invalid {
            value 2;
            description
              "The subscription as a whole is unsupportable with its
              current parameters.";
          }
          enum concluded {
            value 3;
              description
                "A subscription is inactive as it has hit a stop time,
                but not yet been removed from configuration.";
          }
        }
        config false;
        description
          "The presence of this leaf indicates that the subscription
          originated from configuration, not through a control channel
          or RPC.  The value indicates the system established state
          of the subscription.";
      }
```

```
      container receivers {
        description
          "Set of receivers in a subscription.";
        list receiver {
          key "name";
          min-elements 1;
          description
            "A host intended as a recipient for the notification
            messages of a subscription.  For configured subscriptions,
            transport specific network parameters (or a leafref to
            those parameters) may augmentated to a specific receiver
            within this list.";
          leaf name {
            type string;
            description
              "Identifies a unique receiver for a subscription.";
          }
          leaf count-sent {
            type yang:zero-based-counter64;
            config false;
            description
              "The number of event records sent to the receiver.  The
              count is initialized when a dynamic subscription is
              established, or when a configured subscription
              transitions to the valid state.";
          }
          leaf count-excluded {
            type yang:zero-based-counter64;
            config false;
            description
              "The number of event records explicitly removed either
              via an event stream filter or an access control filter so
              that they are not passed to a receiver.  This count is
              set to zero each time 'count-sent' is initialized.";
          }
          leaf state {
            type enumeration {
              enum active {
                value 1;
                description
                  "Receiver is currently being sent any applicable
                  notification messages for the subscription.";
              }
              enum suspended {
                value 2;
                description
                  "Receiver state is 'suspended', so the publisher
                  is currently unable to provide notification messages
```

```
                for the subscription.";
            }
            enum connecting {
              value 3;
              if-feature "configured";
              description
                "A subscription has been configured, but a
                ’subscription-started’ state change notification needs
                to be successfully received before notification
                messages are sent.

                If the ’reset’ action is invoked for a receiver of an
                active configured subscription, the state must be
                moved to ’connecting’.";
            }
            enum timeout {
              value 4;
              if-feature "configured";
              description
                "A subscription has failed in sending a subscription
                started state change to the receiver.
                Additional attempts at connection attempts are not
                currently being made.";
            }
          }
          config false;
          mandatory true;
          description
            "Specifies the state of a subscription from the
            perspective of a particular receiver.  With this info it
            is possible to determine whether a subscriber is currently
            generating notification messages intended for that
            receiver.";
        }
        action reset {
          if-feature "configured";
          description
            "Allows the reset of this configured subscription receiver
            to the ’connecting’ state. This enables the
            connection process to be re-initiated.";
          output {
            leaf time {
              type yang:date-and-time;
              mandatory true;
              description
                "Time a publisher returned the receiver to a
                ’connecting’ state.";
            }
```

```
            }
          }
        }
      }
    }
  }
}
```

<CODE ENDS>

5. Considerations

5.1. IANA Considerations

This document registers the following namespace URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications
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-subscribed-notifications
Namespace: urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications
Prefix: sn
Reference: draft-ietf-netconf-ietf-subscribed-notifications-11.txt (RFC form)

5.2. Implementation Considerations

To support deployments including both configured and dynamic subscriptions, it is recommended to split subscription identifiers into static and dynamic halves. That way it eliminates the possibility of collisions if the configured subscriptions attempt to set a subscription-id which might have already been dynamically allocated. A best practice is to use lower half the "identifier" object’s integer space when that "identifier" is assigned by an external entity (such as with a configured subscription). This leaves the upper half of subscription identifiers available to be dynamically assigned by the publisher.

If a subscription is unable to marshal a series of filtered event records into transmittable notification messages, the receiver should be suspended with the reason "unsupportable-volume".

For configured subscriptions, operations are against the set of receivers using the subscription identifier as a handle for that set. But for streaming updates, state change notifications are local to a receiver. In this specification it is the case that receivers get no information from the publisher about the existence of other receivers. But if a network operator wants to let the receivers correlate results, it is useful to use the subscription identifier across the receivers to allow that correlation.

For configured replay subscriptions, the receiver is protected from duplicated events being pushed after a publisher is rebooted. However it is possible that a receiver might want to acquire event records which failed to be delivered just prior to the reboot. Delivering these event records be accomplished by leveraging the "eventTime" from the last event record received prior to the receipt of a "subscription-started" state change notification. With this "eventTime" and the "replay-start-time" from the "subscription-started" notification, an independent dynamic subscription can be established which retrieves any event records which may have been generated but not sent to the receiver.

5.3. Transport Requirements

This section provides requirements for any subscribed notification transport supporting the solution presented in this document.

For both configured and dynamic subscriptions the publisher MUST authenticate a receiver via some transport level mechanism before sending any event records for which they are authorized to see. In addition, the receiver MUST authenticate the publisher at the transport level. The result is mutual authentication between the two.

A secure transport is highly recommended and the publisher MUST ensure that the receiver has sufficient authorization to perform the function they are requesting against the specific subset of content involved.

A specific transport specification built upon this document may or may not choose to require the use of the same logical channel for the RPCs and the event records. However the event records and the subscription state notifications MUST be sent on the same transport session to ensure the properly ordered delivery.

Additional transport requirements will be dictated by the choice of transport used with a subscription. For an example of such requirements with NETCONF transport, see [I-D.draft-ietf-netconf-netconf-event-notifications].

5.4. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management transports 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 layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF operations and content.

One subscription identifier 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 cannot be assumed that each receiver is getting identical updates.

With configured subscriptions, one or more publishers could be used to overwhelm a receiver. Notification messages SHOULD NOT be sent to any receiver which does not support this specification. Receivers that do not want notification messages need only terminate or refuse any transport sessions from the publisher.

When a receiver of a configured subscription gets a new "subscription-started" message for a known subscription where it is already consuming events, the receiver SHOULD retrieve any event records generated since the last event record was received. This can be accomplish by establishing a separate dynamic replay subscription with the same filtering criteria with the publisher", assuming the publisher supports the "replay" feature.

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 where there is a specific sensitivity/vulnerability:

Container: "/filters"

- o "stream-subtree-filter": updating a filter could increase the computational complexity of all referencing subscriptions.

- o "stream-xpath-filter": updating a filter could increase the computational complexity of all referencing subscriptions.

Container: "/subscriptions"

The following considerations are only relevant for configuration operations made upon configured subscriptions:

- o "configured-replay": can be used to send a large number of event records to a receiver.
- o "dependency": can be used to force important traffic to be queued behind less important updates.
- o "dscp": if unvalidated, can result in the sending of traffic with a higher priority marking than warranted.
- o "identifier": can overwrite an existing subscription, perhaps one configured by another entity.
- o "name": can be used to attempt to send traffic to an unwilling receiver.
- o "replay-start-time": can be used to push very large logs, wasting resources.
- o "source-address": the configured address might not be able to reach a desired receiver.
- o "source-interface": the configured interface might not be able to reach a desired receiver.
- o "source-vrf": can place a subscription into a virtual network where receivers are not entitled to view the subscribed content.
- o "stop-time": could be used to terminate content at an inopportune time.
- o "stream": could set a subscription to an event stream containing no content permitted for the targeted receivers.
- o "stream-filter-ref": could be set to a filter which is irrelevant to the event stream.
- o "stream-subtree-filter": a complex filter can increase the computational resources for this subscription.

- o "stream-xpath-filter": a complex filter can increase the computational resources for this subscription.

- o "weighting": placing a large weight can overwhelm the dequeuing of other subscriptions.

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:

Container: "/streams"

- o "name": if access control is not properly configured, can expose system internals to those who should have no access to this information.

- o "replay-support": if access control is not properly configured, can expose logs to those who should have no access.

Container: "/subscriptions"

- o "count-excluded": leaf can provide information about filtered event records. A network operator should have permissions to know about such filtering.

- o "subscription": different operational teams might have a desire to set varying subsets of subscriptions. Access control should be designed to permit read access to just the allowed set.

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:

RPC: all

- o If a malicious or buggy subscriber sends an unexpectedly large number of RPCs, the result might be an excessive use of system resources on the publisher just to determine that these subscriptions should be declined. In such a situation, subscription interactions MAY be terminated by terminating the transport session.

RPC: "delete-subscription"

- o No special considerations.

RPC: "establish-subscription"

- o Subscriptions could overload a publisher’s resources. For this reason, publishers MUST ensure that they have sufficient resources to fulfill this request or otherwise reject the request.

RPC: "kill-subscription"

- o The "kill-subscription" RPC MUST be secured so that only connections with administrative rights are able to invoke this RPC.

RPC: "modify-subscription"

- o Subscriptions could overload a publisher’s resources. For this reason, publishers MUST ensure that they have sufficient resources to fulfill this request or otherwise reject the request.

6. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Andy Bierman, Tim Jenkins, Martin Bjorklund, Kent Watsen, Balazs Lengyel, Robert Wilton, Sharon Chisholm, Hector Trevino, Susan Hares, Michael Scharf, and Guangying Zheng.

7. References

7.1. Normative References

[I-D.draft-ietf-rtgwg-ni-model]
Berger, L., Hopps, C., and A. Lindem, "YANG Network Instances", draft-ietf-rtgwg-ni-model-12 (work in progress), March 2018.

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

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://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, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://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, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

7.2. Informative References

[I-D.draft-ietf-netconf-netconf-event-notifications] Clemm, Alexander., Voit, Eric., Gonzalez Prieto, Alberto., Nilsen-Nygaard, E., and A. Tripathy, "NETCONF support for event notifications", May 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-netconf-event-notifications/>.

[I-D.draft-ietf-netconf-restconf-notif] Voit, Eric., Clemm, Alexander., Tripathy, A., Nilsen-Nygaard, E., and Alberto. Gonzalez Prieto, "Restconf and HTTP transport for event notifications", May 2018, <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, "YANG Datastore Subscription", May 2018, <https://datatracker.ietf.org/doc/draft-ietf-netconf-yang-push/>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://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, <https://www.rfc-editor.org/info/rfc7923>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Changes between revisions

(To be removed by RFC editor prior to publication)

v13 - v14

- o Removed the ’address’ leaf.
- o Replay is now of type ’empty’ for configured.

v12 - v13

- o Tweaks from Kent’s comments
- o Referenced in YANG model updated per Tom Petch’s comments
- o Added leaf replay-previous-event-time
- o Renamed the event counters, downshifted the subscription states

v11 - v12

- o Tweaks from Kent’s, Tim’s, and Martin’s comments
- o Clarified dscp text, and made its own feature
- o YANG model tweaks alphabetizing, features.

v10 - v11

- o access control filtering of events in streams included to match RFC5277 behavior
- o security considerations updated based on YANG template.
- o dependency QoS made non-normative on HTTP2 QoS
- o tree diagrams referenced for each figure using them
- o reference numbers placed into state machine figures

- o broke configured replay into its own section
- o many tweaks updates based on LC and YANG doctor reviews
- o trees and YANG model reconciled were deltas existed
- o new feature for interface originated.
- o dscp removed from the qos feature
- o YANG model updated in a way which collapses groups only used once so that they are part of the ’subscriptions’ container.
- o alternative encodings only allowed for transports which support them.

v09 - v10

- o Typos and tweaks

v08 - v09

- o NMDA model supported. Non NMDA version at https://github.com/netconf-wg/rfc5277bis/
- o Error mechanism revamped to match to embedded implementations.
- o Explicitly identified error codes relevant to each RPC/Notification

v07 - v08

- o Split YANG trees to separate document subsections.
- o Clarified configured state machine based on Balazs comments, and moved it into the configured subscription subsections.
- o Normative reference to Network Instance model for VRF
- o One transport for all receivers of configured subscriptions.
- o QoS section moved in from yang-push

v06 - v07

- o Clarification on state machine for configured subscriptions.

v05 - v06

- o Made changes proposed by Martin, Kent, and others on the list. Most significant of these are stream returned to string (with the SYSLOG identity removed), intro section on 5277 relationship, an identity set moved to an enumeration, clean up of definitions/ terminology, state machine proposed for configured subscriptions with a clean-up of subscription state options.

- o JSON and XML become features. Also Xpath and subtree filtering become features

- o Terminology updates with event records, and refinement of filters to just event stream filters.

- o Encoding refined in establish-subscription so it takes the RPC's encoding as the default.

- o Namespaces in examples fixed.

v04 - v05

- o Returned to the explicit filter subtyping of v00

- o stream object changed to 'name' from 'stream'

- o Cleaned up examples

- o Clarified that JSON support needs notification-messages draft.

v03 - v04

- o Moved back to the use of RFC5277 one-way notifications and encodings.

v03 - v04

- o Replay updated

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 an event 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 6, 2018

K. Watsen
Juniper Networks
G. Wu
Cisco Systems

June 4, 2018

YANG Groupings for TLS Clients and TLS Servers
draft-ietf-netconf-tls-client-server-06

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:

- I-D.ietf-netconf-trust-anchors

- I-D.ietf-netconf-keystore

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

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

- "YYYY" --> the assigned RFC value for I-D.ietf-netconf-trust-anchors

- "ZZZZ" --> 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:

- "2018-06-04" --> 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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 4
3. The TLS Client Model 4
  3.1. Tree Diagram 4
  3.2. Example Usage 5
  3.3. YANG Module 7
4. The TLS Server Model 10
  4.1. Tree Diagram 10
  4.2. Example Usage 11
  4.3. YANG Module 13

5. The TLS Common Model 16
5.1. Tree Diagram 16
5.2. Example Usage 17
5.3. YANG Module 17
6. Security Considerations 25
7. IANA Considerations 26
7.1. The IETF XML Registry 26
7.2. The YANG Module Names Registry 27
8. Acknowledgements 27
9. References 27
9.1. Normative References 27
9.2. Informative References 29
Appendix A. Change Log 30
A.1. 00 to 01 30
A.2. 01 to 02 30
A.3. 02 to 03 30
A.4. 03 to 04 30
A.5. 04 to 05 31
A.6. 05 to 06 31
Authors' Addresses 31

1. Introduction

This document defines three YANG 1.1 [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 affords 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 "ssh-server-grouping" grouping for the TLS parts it provides, while adding data nodes for the TCP-level call-home configuration.

The modules defined in this document uses groupings defined in [I-D.ietf-netconf-keystore] enabling keys to be either locally defined or a reference to globally configured values.

2. 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.

3. The TLS Client Model

3.1. Tree Diagram

This section provides two tree diagrams [RFC8340] for the "ietf-tls-client" module, the first with used groupings expanded and the second with used groupings not expanded.

The following tree diagram has used groupings expanded:

```
module: ietf-tls-client

  grouping tls-client-grouping
    +-- client-identity
    |  +-- (auth-type)?
    |     +--:(certificate)
    |        +-- certificate
    |           +-- (local-or-keystore)
    |              +--:(local)
    |              |  +-- algorithm
    |              |  |       ct:key-algorithm-ref
    |              |  +-- public-key                binary
    |              |  +-- private-key               union
    |              |  +-- cert
    |              |  |       ct:end-entity-cert-cms
    |              |  +---n certificate-expiration
    |              |     +-- expiration-date?   yang:date-and-time
    |              +--:(keystore) {keystore-implemented}?
    |                 +-- reference
    |                         ks:asymmetric-key-certificate-ref
    +-- server-auth
    |  +-- pinned-ca-certs?       ta:pinned-certificates-ref
    |  +-- pinned-server-certs?   ta:pinned-certificates-ref
    +-- hello-params {tls-client-hello-params-config}?
       +-- tls-versions
       |  +-- tls-version*   identityref
       +-- cipher-suites
          +-- cipher-suite*   identityref
```

The following tree diagram does not have the groupings expanded:

```
[Note: '\' line wrapping for formatting only]

module: ietf-tls-client

  grouping tls-client-grouping
    +-- client-identity
    |  +-- (auth-type)?
    |     +--:(certificate)
    |        +-- certificate
    |           +---u ks:local-or-keystore-end-entity-certificate-gr\
ouping
    +-- server-auth
    |  +-- pinned-ca-certs?       ta:pinned-certificates-ref
    |  +-- pinned-server-certs?   ta:pinned-certificates-ref
    +-- hello-params {tls-client-hello-params-config}?
       +---u tlscmn:hello-params-grouping
```

3.2. Example Usage

This section presents two examples showing the tls-client-grouping populated with some data. These examples are effectively the same except the first configures the client identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 3 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following example configures the client identity using a local key:

[Note: '\' line wrapping for formatting only]

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

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <certificate>
      <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-t\
ypes">ct:rsa1024</algorithm>
      <private-key>base64encodedvalue==</private-key>
      <public-key>base64encodedvalue==</public-key>
      <cert>base64encodedvalue==</cert>
    </certificate>
  </client-identity>

  <!-- which certificates will this client trust -->
  <server-auth>
    <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinned-ca-c\
erts>
    <pinned-server-certs>explicitly-trusted-server-certs</pinned-ser\
ver-certs>
  </server-auth>

</tls-client>
```

The following example configures the client identity using a key from the keystore:

```
[Note: '\' line wrapping for formatting only]

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

  <!-- how this client will authenticate itself to the server -->
  <client-identity>
    <certificate>
      <reference>ex-rsa-cert</reference>
    </certificate>
  </client-identity>

  <!-- which certificates will this client trust -->
  <server-auth>
    <pinned-ca-certs>explicitly-trusted-server-ca-certs</pinned-ca-c\
erts>
    <pinned-server-certs>explicitly-trusted-server-certs</pinned-ser\
ver-certs>
  </server-auth>

</tls-client>
```

3.3. YANG Module

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

```
<CODE BEGINS> file "ietf-tls-client@2018-06-04.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 tlscmn;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
  }

  import ietf-trust-anchors {
    prefix ta;
    reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
  }

  import ietf-keystore {
    prefix ks;
```

```
       reference
         "RFC ZZZZ: YANG Data Model for a 'Keystore' Mechanism";
     }

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

     contact
      "WG Web:   <http://datatracker.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 TLS client that
       can be used as a basis for specific TLS client instances.

       Copyright (c) 2018 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 "2018-06-04" {
       description
        "Initial version";
       reference
        "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
     }

     // features

     feature tls-client-hello-params-config {
       description
         "TLS hello message parameters are configurable on a TLS
          client.";
```

```
     }

     // groupings

     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 client-identity {
         description
           "The credentials used by the client to authenticate to
            the TLS server.";

         choice auth-type {
           description
             "The authentication type.";
           container certificate {
             uses ks:local-or-keystore-end-entity-certificate-grouping;
             description
               "A locally-defined or referenced certificate
                to be used for client authentication.";
             reference
               "RFC ZZZZ: YANG Data Model for a 'Keystore' Mechanism";
           }
         }
       } // end client-identity

       container server-auth {
         must 'pinned-ca-certs or pinned-server-certs';
         description
           "Trusted server identities.";
         leaf pinned-ca-certs {
           type ta:pinned-certificates-ref;
           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 pinned CA certificate.";
         }

         leaf pinned-server-certs {
           type ta:pinned-certificates-ref;
           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 pinned server certificate.";
          }
        }

        container hello-params {
          if-feature tls-client-hello-params-config;
          uses tlscmn:hello-params-grouping;
          description
            "Configurable parameters for the TLS hello message.";
        }

      } // end tls-client-grouping

    }
    <CODE ENDS>
```

4. The TLS Server Model

4.1. Tree Diagram

This section provides two tree diagrams [RFC8340] for the "ietf-tls-server" module, the first with used groupings expanded and the second with used groupings not expanded.

The following tree diagram has used groupings expanded:

```
module: ietf-tls-server

  grouping tls-server-grouping
    +-- server-identity
    |  +-- (local-or-keystore)
    |     +--:(local)
    |     |  +-- algorithm                ct:key-algorithm-ref
    |     |  +-- public-key               binary
    |     |  +-- private-key              union
    |     |  +-- cert                     ct:end-entity-cert-cms
    |     |  +---n certificate-expiration
    |     |     +-- expiration-date?   yang:date-and-time
    |     +--:(keystore) {keystore-implemented}?
    |        +-- reference
    |                ks:asymmetric-key-certificate-ref
    +-- client-auth
    |  +-- pinned-ca-certs?       ta:pinned-certificates-ref
    |  +-- pinned-client-certs?   ta:pinned-certificates-ref
    +-- hello-params {tls-server-hello-params-config}?
       +-- tls-versions
       |  +-- tls-version*   identityref
       +-- cipher-suites
          +-- cipher-suite*   identityref
```

The following tree diagram does not have the used groupings expanded:

```
module: ietf-tls-server

  grouping tls-server-grouping
    +-- server-identity
    |  +---u ks:local-or-keystore-end-entity-certificate-grouping
    +-- client-auth
    |  +-- pinned-ca-certs?       ta:pinned-certificates-ref
    |  +-- pinned-client-certs?   ta:pinned-certificates-ref
    +-- hello-params {tls-server-hello-params-config}?
       +---u tlscmn:hello-params-grouping
```

4.2. Example Usage

This section presents two examples showing the tls-server-grouping populated with some data. These examples are effectively the same except the first configures the server identity using a local key while the second uses a key configured in a keystore. Both examples are consistent with the examples presented in Section 3 of [I-D.ietf-netconf-trust-anchors] and Section 3.2 of [I-D.ietf-netconf-keystore].

The following example configures the server identity using a local key:

[Note: '\' line wrapping for formatting only]

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

  <!-- how this server will authenticate itself to the client -->
  <server-identity>
    <algorithm xmlns:ct="urn:ietf:params:xml:ns:yang:ietf-crypto-typ\
es">ct:rsa1024</algorithm>
    <private-key>base64encodedvalue==</private-key>
    <public-key>base64encodedvalue==</public-key>
    <cert>base64encodedvalue==</cert>
  </server-identity>

  <!-- which certificates will this server trust -->
  <client-auth>
    <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-c\
erts>
    <pinned-client-certs>explicitly-trusted-client-certs</pinned-cli\
ent-certs>
  </client-auth>

</tls-server>
```

The following example configures the server identity using a key from the keystore:

```
[Note: '\' line wrapping for formatting only]

<tls-server xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-server">

  <!-- how this server will authenticate itself to the client -->
  <server-identity>
    <reference>ex-rsa-cert</reference>
  </server-identity>

  <!-- which certificates will this server trust -->
  <client-auth>
    <pinned-ca-certs>explicitly-trusted-client-ca-certs</pinned-ca-c\
erts>
    <pinned-client-certs>explicitly-trusted-client-certs</pinned-cli\
ent-certs>
  </client-auth>

</tls-server>
```

4.3. YANG Module

This YANG module has a normative references to [RFC5246], [I-D.ietf-netconf-trust-anchors] and [I-D.ietf-netconf-keystore].

```
<CODE BEGINS> file "ietf-tls-server@2018-06-04.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 tlscmn;
    revision-date 2018-06-04; // stable grouping definitions
    reference
      "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
  }

  import ietf-trust-anchors {
    prefix ta;
    reference
      "RFC YYYY: YANG Data Model for Global Trust Anchors";
  }

  import ietf-keystore {
    prefix ks;
    reference
      "RFC ZZZZ: YANG Data Model for a 'Keystore' Mechanism";
```

```
     }

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

     contact
      "WG Web:   <http://datatracker.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 TLS server that
       can be used as a basis for specific TLS server instances.

       Copyright (c) 2018 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 "2018-06-04" {
       description
        "Initial version";
       reference
        "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
     }

     // features

     feature tls-server-hello-params-config {
       description
         "TLS hello message parameters are configurable on a TLS
          server.";
     }
```

```
     // groupings

     grouping tls-server-grouping {
       description
         "A reusable grouping for configuring a TLS server without
          any consideration for how underlying TCP sessions are
          established.";

       container server-identity {
         description
           "A locally-defined or referenced end-entity certificate,
            including any configured intermediate 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
            RFC ZZZZ:
              YANG Data Model for a 'Keystore' Mechanism";
         uses ks:local-or-keystore-end-entity-certificate-grouping;
       }

       container client-auth {
         description
           "A reference to a list of pinned certificate authority (CA)
            certificates and a reference to a list of pinned client
            certificates.";
         leaf pinned-ca-certs {
           type ta:pinned-certificates-ref;
           description
             "A reference to a list of certificate authority (CA)
              certificates used by the TLS server to authenticate
              TLS client certificates.  A client certificate is
              authenticated if it has a valid chain of trust to
              a configured pinned CA certificate.";
           reference
             "RFC YYYY: YANG Data Model for Global Trust Anchors";
         }
         leaf pinned-client-certs {
           type ta:pinned-certificates-ref;
           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 pinned client certificate.";
           reference
             "RFC YYYY: YANG Data Model for Global Trust Anchors";
```

```
        }
      }

      container hello-params {
        if-feature tls-server-hello-params-config;
        uses tlscmn:hello-params-grouping;
        description
          "Configurable parameters for the TLS hello message.";
      }

    } // end tls-server-grouping

  }
  <CODE ENDS>
```

5. 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.

5.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model for the "ietf-tls-common" module.

```
module: ietf-tls-common

  grouping hello-params-grouping
    +-- tls-versions
    |  +-- tls-version*   identityref
    +-- cipher-suites
       +-- cipher-suite*   identityref
```

5.2. Example Usage

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

```
<hello-params
   xmlns="urn:ietf:params:xml:ns:yang:ietf-tls-common"
   xmlns:tlscmn="urn:ietf:params:xml:ns:yang:ietf-tls-common">
  <tls-versions>
    <tls-version>tlscmn:tls-1.1</tls-version>
    <tls-version>tlscmn:tls-1.2</tls-version>
  </tls-versions>
  <cipher-suites>
    <cipher-suite>tlscmn:dhe-rsa-with-aes-128-cbc-sha</cipher-suite>
    <cipher-suite>tlscmn:rsa-with-aes-128-cbc-sha</cipher-suite>
    <cipher-suite>tlscmn:rsa-with-3des-ede-cbc-sha</cipher-suite>
  </cipher-suites>
</hello-params>
```

5.3. YANG Module

This YANG module has a normative references to [RFC2246], [RFC4346], [RFC4492], [RFC5246], [RFC5288], and [RFC5289].

```
<CODE BEGINS> file "ietf-tls-common@2018-06-04.yang"
module ietf-tls-common {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-tls-common";
  prefix "tlscmn";

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

  contact
   "WG Web:   <http://datatracker.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) 2018 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 "2018-06-04" {
       description
        "Initial version";
       reference
        "RFC XXXX: YANG Groupings for TLS Clients and TLS Servers";
     }

     // features

     feature tls-1_0 {
       description
         "TLS Protocol Version 1.0 is supported.";
       reference
         "RFC 2246: The TLS Protocol Version 1.0";
     }

     feature tls-1_1 {
       description
         "TLS Protocol Version 1.1 is supported.";
       reference
         "RFC 4346: The Transport Layer Security (TLS) Protocol
                    Version 1.1";
     }

     feature tls-1_2 {
       description
         "TLS Protocol Version 1.2 is supported.";
       reference
         "RFC 5246: The Transport Layer Security (TLS) Protocol
                    Version 1.2";
     }

     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-1.0 {
      base tls-version-base;
      if-feature tls-1_0;
```

```
      description
        "TLS Protocol Version 1.0.";
      reference
        "RFC 2246: The TLS Protocol Version 1.0";
    }

    identity tls-1.1 {
      base tls-version-base;
      if-feature tls-1_1;
      description
        "TLS Protocol Version 1.1.";
      reference
        "RFC 4346: The Transport Layer Security (TLS) Protocol
                   Version 1.1";
    }

    identity tls-1.2 {
      base tls-version-base;
      if-feature tls-1_2;
      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-ede-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.";
      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
              "Acceptable TLS protocol versions.

               If this leaf-list is not configured (has zero elements)
               the acceptable TLS protocol versions are implementation-
               defined.";
          }
        }
        container cipher-suites {
          description
            "Parameters regarding cipher suites.";
          leaf-list cipher-suite {
            type identityref {
              base cipher-suite-base;
            }
            ordered-by user;
            description
              "Acceptable cipher suites in order of descending
               preference.

               If this leaf-list is not configured (has zero elements)
               the acceptable cipher suites are implementation-
               defined.";
          }
        }

      } // end hello-params-grouping

    }
    <CODE ENDS>
```

6. Security Considerations

The YANG modules defined in this document are 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.

Since the modules defined in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

There are a number of data nodes defined in the YANG modules 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 of all the groupings defined in this draft 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 the YANG modules 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

7. IANA Considerations

7.1. 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.
```

7.2. 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:       tlscmn
reference:    RFC XXXX
```

8. 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.

9. References

9.1. Normative References

[I-D.ietf-netconf-keystore]
Watsen, K., "YANG Data Model for a "Keystore" Mechanism", draft-ietf-netconf-keystore-04 (work in progress), October 2017.

[I-D.ietf-netconf-trust-anchors]
Watsen, K., "YANG Data Model for Global Trust Anchors", draft-ietf-netconf-trust-anchors-00 (work in progress), June 2018.

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

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, DOI 10.17487/RFC2246, January 1999, <https://www.rfc-editor.org/info/rfc2246>.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, DOI 10.17487/RFC4346, April 2006, <https://www.rfc-editor.org/info/rfc4346>.

[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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6536>.

[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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc8174>.

9.2. Informative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Change Log

A.1. 00 to 01

- Noted that ’0.0.0.0’ and ’::’ might have special meanings.
- Renamed "keychain" to "keystore".

A.2. 01 to 02

- Removed the groupings containing transport-level configuration. Now modules contain only the transport-independent groupings.
- Filled in previously incomplete ’ietf-tls-client’ module.
- Added cipher suites for various algorithms into new ’ietf-tls-common’ module.

A.3. 02 to 03

- Added a ’must’ statement to container ’server-auth’ asserting that at least one of the various auth mechanisms must be specified.
- Fixed description statement for leaf ’trusted-ca-certs’.

A.4. 03 to 04

- Updated title to "YANG Groupings for TLS Clients and TLS Servers"
- Updated leafref paths to point to new keystore path
- Changed the YANG prefix for ietf-tls-common from ’tlscom’ to ’tlscmn’.
- Added TLS protocol verions 1.0 and 1.1.
- Made author lists consistent
- Now tree diagrams reference ietf-netmod-yang-tree-diagrams
- Updated YANG to use typedefs around leafrefs to common keystore paths
- Now inlines key and certificates (no longer a leafref to keystore)

A.5. 04 to 05

- o Merged changes from co-author.

A.6. 05 to 06

- o Updated to use trust anchors from trust-anchors draft (was keystore draft)
- o Now Uses new keystore grouping enabling asymmetric key to be either locally defined or a reference to the keystore.

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Gary Wu
Cisco Systems

EMail: garywu@cisco.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 6, 2018

K. Watsen
Juniper Networks
June 4, 2018

YANG Data Model for Global Trust Anchors
draft-ietf-netconf-trust-anchors-00

Abstract

This document defines a YANG 1.1 data model for configuring global sets of X.509 certificates and SSH host-keys that can be referenced by other data models for trust. While the SSH host-keys are uniquely for the SSH protocol, the X.509 certificates may have multiple uses, including authenticating protocol peers and verifying signatures.

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:

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

- "YYYY" --> the assigned RFC value for draft-ietf-netconf-crypto-types

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

- "2018-06-04" --> the publication date of this draft

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

- 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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 6, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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 Trust Anchors Model 3
  2.1. Tree Diagram 3
  2.2. Example Usage 4
  2.3. YANG Module 7
3. Security Considerations 12
4. IANA Considerations 12
  4.1. The IETF XML Registry 12
  4.2. The YANG Module Names Registry 13
5. References 13
  5.1. Normative References 13
  5.2. Informative References 13
Appendix A. Change Log 15
  A.1. I-D to 00 15
Acknowledgements 15
Author's Address 15

1. Introduction

This document defines a YANG 1.1 [RFC7950] data model for configuring global sets of X.509 certificates and SSH host-keys that can be referenced by other data models for trust. While the SSH host-keys are uniquely for the SSH protocol, the X.509 certificates may be used for multiple uses, including authenticating protocol peers and verifying signatures.

This document in compliant with Network Management Datastore Architecture (NMDA) [RFC8342]. For instance, to support trust anchors installed during manufacturing, it is expected that such data may appear only in <operational>.

1.1. 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.

1.2. Tree Diagram Notation

Tree diagrams used in this document follow the notation defined in [RFC8340].

2. The Trust Anchors Model

2.1. Tree Diagram

The following tree diagram provides an overview of the "ietf-trust-anchors" module.

```
module: ietf-trust-anchors
  +--rw trust-anchors
     +--rw pinned-certificates* [name]
     |  +--rw name                 string
     |  +--rw description?         string
     |  +--rw pinned-certificate* [name]
     |     +--rw name                     string
     |     +--rw cert                     ct:trust-anchor-cert-cms
     |     +---n certificate-expiration
     |        +-- expiration-date?   yang:date-and-time
     +--rw pinned-host-keys* [name]
        +--rw name               string
        +--rw description?       string
        +--rw pinned-host-key* [name]
           +--rw name        string
           +--rw host-key    ct:ssh-host-key
```

2.2. Example Usage

The following example illustrates trust anchors in <operational> as described by Section 5.3 in [RFC8342]. This datastore view illustrates data set by the manufacturing process alongside conventional configuration. This trust anchors instance has five sets of pinned certificates and one set of pinned host keys.

```
<trust-anchors
   xmlns="urn:ietf:params:xml:ns:yang:ietf-trust-anchors"
   xmlns:or="urn:ietf:params:xml:ns:yang:ietf-origin">

  <!-- Manufacturer’s trusted root CA certs -->
  <pinned-certificates or:origin="or:system">
    <name>manufacturers-root-ca-certs</name>
    <description>
      Certificates built into the device for authenticating
      manufacturer-signed objects, such as TLS server certificates,
      vouchers, etc.  Note, though listed here, these are not
      configurable; any attempt to do so will be denied.
    </description>
    <pinned-certificate>
      <name>Manufacturer Root CA cert 1</name>
      <cert>base64encodedvalue==</cert>
    </pinned-certificate>
    <pinned-certificate>
      <name>Manufacturer Root CA cert 2</name>
      <cert>base64encodedvalue==</cert>
    </pinned-certificate>
  </pinned-certificates>
```

```
<!-- specific end-entity certs for authenticating servers -->
<pinned-certificates or:origin="or:intended">
  <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 pinned CA.
  </description>
  <pinned-certificate>
    <name>Fred Flintstone</name>
    <cert>base64encodedvalue==</cert>
  </pinned-certificate>
</pinned-certificates>

<!-- trusted CA certs for authenticating servers -->
<pinned-certificates or:origin="or:intended">
  <name>explicitly-trusted-server-ca-certs</name>
  <description>
    Trust anchors (i.e. CA certs) that are used to authenticate
    server connections.  Servers are authenticated if their
    certificate has a chain of trust to one of these CA
    certificates.
  </description>
  <pinned-certificate>
    <name>ca.example.com</name>
    <cert>base64encodedvalue==</cert>
  </pinned-certificate>
</pinned-certificates>

<!-- specific end-entity certs for authenticating clients -->
<pinned-certificates or:origin="or:intended">
  <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 pinned CA.
  </description>
  <pinned-certificate>
    <name>George Jetson</name>
    <cert>base64encodedvalue==</cert>
  </pinned-certificate>
</pinned-certificates>

<!-- trusted CA certs for authenticating clients -->
<pinned-certificates or:origin="or:intended">
  <name>explicitly-trusted-client-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 CA
        certificates.
      </description>
      <pinned-certificate>
        <name>ca.example.com</name>
        <cert>base64encodedvalue==</cert>
      </pinned-certificate>
    </pinned-certificates>

    <!-- trusted CA certs for random HTTPS servers on Internet -->
    <pinned-certificates or:origin="or:system">
      <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>
      <pinned-certificate>
        <name>ex-certificate-authority</name>
        <cert>base64encodedvalue==</cert>
      </pinned-certificate>
    </pinned-certificates>

    <!-- specific SSH host keys for authenticating clients -->
    <pinned-host-keys or:origin="or:intended">
      <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>
      <pinned-host-key>
        <name>corp-fw1</name>
        <host-key>base64encodedvalue==</host-key>
      </pinned-host-key>
    </pinned-host-keys>

  </trust-anchors>
```

The following example illustrates the "certificate-expiration" notification in use with the NETCONF protocol.

[Note: '\' line wrapping for formatting only]

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2018-05-25T00:01:00Z</eventTime>
  <trust-anchors
    xmlns="urn:ietf:params:xml:ns:yang:ietf-trust-anchors">
    <pinned-certificates>
      <name>explicitly-trusted-client-certs</name>
      <pinned-certificate>
        <name>George Jetson</name>
        <certificate-expiration>
          <expiration-date>2018-08-05T14:18:53-05:00</expiration-dat\
e>
        </certificate-expiration>
      </pinned-certificate>
    </pinned-certificates>
  </trust-anchors>
</notification>
```

2.3. YANG Module

This YANG module imports modules from [RFC6536], [RFC6991] and [I-D.ietf-netconf-crypto-types].

```
  <CODE BEGINS> file "ietf-trust-anchors@2018-06-04.yang"
module ietf-trust-anchors {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-trust-anchors";
  prefix "ta";

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

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC YYYY: Common YANG Data Types for Cryptography";
  }

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

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

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


     description
      "This module defines a data model for configuring global
       trust anchors used by other data models.  The data model
       enables the configuration of sets of trust anchors.
       This data model supports configuring trust anchors for
       both X.509 certificates and SSH host keys.

       Copyright (c) 2018 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 "2018-06-04" {
       description
        "Initial version";
       reference
        "RFC XXXX: YANG Data Model for Global Trust Anchors";
     }


     /*************************************************************/
     /*   Typedefs for leafrefs to commonly referenced objects   */
     /*************************************************************/

     typedef pinned-certificates-ref {
       type leafref {
         path "/ta:trust-anchors/ta:pinned-certificates/ta:name";
         require-instance false;
       }
       description
         "This typedef enables importing modules to easily define a
          leafref to a ’pinned-certificates’ object.  The require
```

```
          instance attribute is false to enable the referencing of
          pinned certificates that exist only in <operational>.";
       reference
         "RFC 8342: Network Management Datastore Architecture (NMDA)";
     }

     typedef pinned-host-keys-ref {
       type leafref {
         path "/ta:trust-anchors/ta:pinned-host-keys/ta:name";
         require-instance false;
       }
       description
         "This typedef enables importing modules to easily define a
          leafref to a 'pinned-host-keys' object.  The require
          instance attribute is false to enable the referencing of
          pinned host keys that exist only in <operational>.";
       reference
         "RFC 8342: Network Management Datastore Architecture (NMDA)";
     }


     /**********************************/
     /*   Protocol accessible nodes   */
     /**********************************/

     container trust-anchors {
       description
         "Contains sets of X.509 certificates and SSH host keys.";

       list pinned-certificates {
         key name;
         description
           "A list of pinned certificates.  These certificates can be
            used by a server to authenticate clients, or by a client
            to authenticate servers.  Each list of pinned certificates
            SHOULD be specific to a purpose, as the list as a whole
            may be referenced by other modules.  For instance, a
            NETCONF server's configuration might use a specific list
            of pinned certificates for when authenticating NETCONF
            client connections.";
         leaf name {
           type string;
           description
             "An arbitrary name for this list of pinned
              certificates.";
         }
         leaf description {
           type string;
```

```
          description
            "An arbitrary description for this list of pinned
             certificates.";
        }
        list pinned-certificate {
          key name;
          description
            "A pinned certificate.";
          leaf name {
            type string;
            description
              "An arbitrary name for this pinned certificate. The
               name must be unique across all lists of pinned
               certificates (not just this list) so that leafrefs
               from another module can resolve to unique values.";
          }
          leaf cert {
            type ct:trust-anchor-cert-cms;
            mandatory true;
            description
              "The binary certificate data for this pinned
               certificate.";
            reference
              "RFC YYYY: Common YANG Data Types for Cryptography";
          }
          notification certificate-expiration {
            description
             "A notification indicating that the configured trust
              anchor 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 until the
              issue is resolved.";
            leaf expiration-date {
              type yang:date-and-time;
              //mandatory true;
              description
                "Identifies the expiration date on the certificate.";
            }
          }

        }
      }
      list pinned-host-keys {
        key name;
        description
          "A list of pinned host keys.  These pinned host-keys can
```

```
            be used by clients to authenticate SSH servers.  Each
            list of pinned host keys SHOULD be specific to a purpose,
            so the list as a whole may be referenced by other modules.
            For instance, a NETCONF client's configuration might
            point to a specific list of pinned host keys for when
            authenticating specific SSH servers.";
         leaf name {
           type string;
           description
             "An arbitrary name for this list of pinned SSH
              host keys.";
         }
         leaf description {
           type string;
           description
             "An arbitrary description for this list of pinned SSH
              host keys.";
         }
         list pinned-host-key {
           key name;
           description
             "A pinned host key.";
           leaf name {
             type string;
             description
               "An arbitrary name for this pinned host-key. Must be
                unique across all lists of pinned host-keys (not just
                this list) so that a leafref to it from another module
                can resolve to unique values.";
           }
           leaf host-key {
             type ct:ssh-host-key;
             mandatory true;
             description
               "The binary public key data for this pinned host key.";
             reference
               "RFC YYYY: Common YANG Data Types for Cryptography";
           }
         }
       }
     }

   }
     <CODE ENDS>
```

3. 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 any trust anchor may 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’.

None of the readable data nodes in this YANG module are considered sensitive or vulnerable in network environments.

This module does not define any RPCs, actions, or notifications, and thus the security consideration for such is not provided here.

4. IANA Considerations

4.1. The IETF XML Registry

This document registers one URI in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in [RFC3688], the following registration is requested:

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

4.2. 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-trust-anchors
namespace:    urn:ietf:params:xml:ns:yang:ietf-trust-anchors
prefix:       ta
reference:    RFC XXXX
```

5. References

5.1. Normative References

[I-D.ietf-netconf-crypto-types]
Watsen, K., "Common YANG Data Types for Cryptography", draft-ietf-netconf-crypto-types-00 (work in progress), June 2018.

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

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, DOI 10.17487/RFC6536, March 2012, <https://www.rfc-editor.org/info/rfc6536>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc8174>.

5.2. Informative References

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

Appendix A. Change Log

A.1. I-D to 00

- Now imports and uses the crypto-types module.
- FIXME
- FIXME
- FIXME: added notification example...

Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by last name): Martin Bjorklund, Balazs Kovacs, Eric Voit, and Liang Xia.

Author’s Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: January 1, 2019

G. Zheng
T. Zhou
A. Clemm
Huawei

June 30, 2018

UDP based Publication Channel for Streaming Telemetry
draft-ietf-netconf-udp-pub-channel-03

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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on January 1, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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. Transport Mechanisms 5
4.1. Dynamic Subscription 5
4.2. Configured Subscription 7
5. UDP Transport for Publication Channel 8
5.1. Design Overview 8
5.2. Data Format of the UPC Message Header 8
5.3. Options 10
5.3.1. Reliability Option 10
5.3.2. Fragmentation Option 11
5.4. Data Encoding 12
6. Using DTLS to Secure UPC 12
6.1. Transport 12
6.2. Port Assignment 13
6.3. DTLS Session Initiation 13
6.4. Sending Data 14
6.5. Closure 14
7. Congestion Control 15
8. IANA Considerations 15
9. Security Considerations 16
10. Acknowledgements 16
11. References 16
11.1. Normative References 16
11.2. Informative References 17
11.3. URIs 18
Appendix A. Change Log 18
Authors' Addresses 19

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] 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, NETCONF transport [I-D.ietf-netconf-netconf-event-notifications] and HTTP transport [I-D.ietf-netconf-restconf-notif], have been defined so far for the notification messages.

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. The transports that have been defined so far, NETCONF and HTTP, are ultimately based on TCP and lack the efficiency needed to stream data continuously at high velocity. A lighter-weight, more efficient transport, e.g. a transport based on UDP is needed.

- o Firstly, data collector will suffer a lot of TCP connections from, for example, many line cards equipped on different devices.

- o Secondly, as no connection state needs to be maintained, UDP encapsulation can be easily implemented by hardware which will further improve the performance.

- o Thirdly, because of the lightweight UDP encapsulation, higher frequency and better transit performance can be achieved, which is important for streaming telemetry.

This document specifies a higher-performance transport option for Sub-Notif that leverages UDP. Specifically, it facilitates the distributed data collection mechanism described in [I-D.zhou-netconf-multi-stream-originators]. In the case of data originating from multiple line cards, the centralized 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. The centralized data collection mechanism can result in a performance bottleneck, especially when large amounts of data are involved. What is needed instead is the 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. The proposed UDP based Publication Channel (UPC) natively supports the distributed data collection mechanism.

The transport described in this document can be used for transmitting notification messages over both IPv4 and IPv6 [RFC8200].

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-subscribed-notifications] with extensions [I-D.zhou-netconf-multi-stream-originators].

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 telemetry source which is managed and controlled by a single Subscription Server.

Component Subscription Server: 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 Fig. 1. Both the Collector and the Publisher can be distributed. The Collector includes the Subscriber and a set of Receivers. And the Publisher includes a Subscription Server and a set of Component Subscription Servers. The Subscriber cannot see the Component Subscription Servers directly, so it will send the Global Subscription information to the Subscription Server (e.g., main board) via the Subscription Channel. When receiving a Global Subscription, the Subscription Server decomposes the subscription request into multiple Component Subscriptions, each involving data from a separate internal telemetry source, for example a line card. The Component Subscriptions are distributed to the Component Subscription Server. Subsequently, each data originator generates its own stream of telemetry data, collecting and encapsulating the packets per the Component Subscription and streaming them to the designated Receivers. This distributed data collection mechanism may form multiple Publication Channels to the Receivers. The Receiver is able to assemble many pieces of data associated with one Global Subscription.

The Publication Channel supports the reliable data streaming, for example for some alarm events. The Collector has the option of deducing the packet loss and the disorder based on the information carried by the notification data. And the Collector may decide the behavior to request retransmission.

The rest of the draft describes the UDP based Publication Channel (UPC).

```
                  +-------------------------------------+
                  |            Collector                |
                  |                                     |
                  |  +------------+  +-----------+     |
                  |  | Subscriber |  | Receivers |     |
                  |  +----+-------+  +--^----^---+     |
                  |       |             |    |          |
                  +-------------------------------------+
                          |             |    |
             Subscription |             |    | Publication
             Channel      |             |    | Channel
                          |   +---------+    |
                          |   |              |
                  +-------------------------------------+
                  |       |   |              |          |
                  |  +----v---+-----+ +------+-------+  |
                  |  | Subscription | | Component    |  |
                  |  | Server       | | Subscription |  |
                  |  |              | | Servers      |  |
                  |  +--------------+ +--------------+  |
                  |                                     |
                  |              Publisher              |
                  +-------------------------------------+
```

Fig. 1 Distributed Data Collection

4. Transport Mechanisms

For a complete pub-sub mechanism, this section will describe how the UPC is used to interact with the Subscription Channel relying on NETCONF or RESTCONF.

4.1. Dynamic Subscription

Dynamic subscriptions for Sub-Notif are configured and managed via signaling messages transported over NETCONF [RFC6241] or RESTCONF [RFC8040]. The Sub-Notif defined RPCs which are sent and responded via the Subscription Channel (a), between the Subscriber and the Subscription Server of the Publisher. In this case, only one

Receiver is associated with the Subscriber. In the Publisher, there may be multiple data originators. Notification messages are pushed on separate channels (b), from different data originators to the Receiver.

```
+--------------+                        +--------------+
|  Collector   |                        |  Publisher   |
|              |                        |              |
|  (a)   (b)   |                        |  (a)    (b)  |
+--+------+----+                        +--+-------+---+
   |      |                                |       |
   |      |     RPC:establish-subscription |       |
   +----------------------------------------->       |
   |      |     RPC Reply: OK              |       |
   <----------------------------------------+       |
   |      |     UPC:notifications          |       |
   |      <----------------------------------------+
   |      |                                |       |
   |      |     RPC:modify-subscription    |       |
   +----------------------------------------->       |
   |      |     RPC Reply: OK              |       |
   <----------------------------------------+       |
   |      |     UPC:notifications          |       |
   |      <----------------------------------------+
   |      |                                |       |
   |      |     RPC:delete-subscription    |       |
   +----------------------------------------->       |
   |      |     RPC Reply: OK              |       |
   <----------------------------------------+       |
   |      |                                |       |
   |      |                                |       |
   +      +                                +       +
```

Fig. 2 Call Flow For Dynamic Subscription

In the case of dynamic subscription, the Receiver and the Subscriber SHOULD be colocated. So UPC can use the source IP address of the Subscription Channel as it’s destination IP address. The Receiver MUST support listening messages at the IANA-assigned PORT-X or PORT-Y, but MAY be configured to listen at a different port.

The Publication Channels MUST share fate with the subscription session. In other words, when the delete-subscription is received or the subscription session is broken, all the associated Publication Channels MUST be closed.

4.2. Configured Subscription

For a Configured Subscription, there is no guarantee that the Subscriber is currently in place with the associated Receiver(s). As defined in Sub-Notif, the subscription configuration contains the location information of all the receivers, including the IP address and the port number. So that the data originator can actively send generated messages to the corresponding Receivers via the UPC.

The first message MUST be a separate subscription-started notification to indicate the Receiver that the pushing is started. Then, the notifications can be sent immediately without any wait.

All the subscription state notifications, as defined in [I-D.ietf-netconf-subscribed-notifications], MUST be encapsulated to be separated notification messages.

```
+--------------+                        +--------------+
|  Collector   |                        |  Publisher   |
|              |                        |              |
|  (a)   (b)   |                        |  (a)    (b)  |
+--+------+----+                        +--+-------+---+
   |      |                                |       |
   |      |     Capability Exchange        |       |
   <----------------------------------------->      |
   |      |                                |       |
   |      |     Edit config(create)        |       |
   +----------------------------------------->      |
   |      |     RPC Reply: OK              |       |
   <----------------------------------------+       |
   |      |     UPC:subscription started   |       |
   |      <----------------------------------------+
   |      |     UPC:notifications          |       |
   |      <----------------------------------------+
   |      |                                |       |
   |      |     Edit config(delete)        |       |
   +----------------------------------------->      |
   |      |     RPC Reply: OK              |       |
   <----------------------------------------+       |
   |      |     UPC:subscription terminated |       |
   |      <----------------------------------------+
   |      |                                |       |
   |      |                                |       |
   +      +                                +       +
```

Fig. 3 Call Flow For Configured Subscription

5. UDP Transport for Publication Channel

5.1. Design Overview

As specified in Sub-Notif, the telemetry data is encapsulated in the NETCONF/RESTCONF notification message, which is then encapsulated and carried in the transport protocols, e.g. TLS, HTTP2. The following figure shows the overview of the typical UPC message structure.

- o The Message Header contains information that can facilitate the message transmission before de-serializing the notification message.

- o Notification Message is the encoded content that the publication channel transports. The common encoding method includes GPB [1], CBOR [RFC7049], JSON, and XML. [I-D.ietf-netconf-notification-messages] describes the structure of the Notification Message for both single notification and multiple bundled notifications.

```
+-------+  +--------------+  +--------------+
|  UDP  |  |   Message    |  | Notification |
|       |  |   Header     |  | Message      |
+-------+  +--------------+  +--------------+
```

Fig. 4 UDP Publication Message Overview

5.2. Data Format of the UPC Message Header

The UPC Message Header contains information that can facilitate the message transmission before de-serializing the notification message. The data format is shown as follows.

```
 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
+-------+---------------+-------+-------------------------------+
| Vers. |    Flag       |  ET   |      Length                   |
+-------+---------------+-------+-------------------------------+
|                       Message-Generator-ID                    |
+---------------------------------------------------------------+
|                       Message ID                              |
+---------------------------------------------------------------+
~                       Options                                 ~
+---------------------------------------------------------------+
```

Fig. 3 UPC Message Header Format

The Message Header contains the following field:

- o Vers.: represents the PDU (Protocol Data Unit) encoding version. The initial version value is 0.

- o Flag: is a bitmap indicating what features this packet has and the corresponding options attached. Each bit associates to one feature and one option data. When the bit is set to 1, the associated feature is enabled and the option data is attached. The sequence of the presence of the options follows the bit order of the bitmap. In this document, the flag is specified as follows:

  - * bit 0, the reliability flag;

  - * bit 1, the fragmentation flag;

  - * other bits are reserved.

- o ET: is a 4 bits identifier to indicate the encoding type used for the Notification Message. 16 types of encoding can be expressed:

  - * 0: GPB;

  - * 1: CBOR;

  - * 2: JSON;

  - * 3: XML;

  - * others are reserved.

- o Length: is the total length of the message, measured in octets, including message header.

- o Message-Generator-ID: is a 32-bit identifier of the process which created the notification message. This allows disambiguation of an information source, such as the identification of different line cards sending the notification messages. The source IP address of the UDP datagrams SHOULD NOT be interpreted as the identifier for the host that originated the UPC message. The entity sending the UPC message could be merely a relay.

- o The Message ID is generated continuously by the message generator. Different subscribers share the same notification ID sequence.

- o Options: is a variable-length field. The details of the Options will be described in the respective sections below.

5.3. Options

The order of packing the data fields in the Options field follows the bit order of the Flag field.

5.3.1. Reliability Option

The UDP based publication transport described in this document provides two streaming modes, the reliable mode an 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 operate need to consider the reliable requirement together with the resource usage.

When the reliability flag bit is set to 1 in the Flag field, the following option data will be attached

```
 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
+-----------------------------------------------------------------+
|            Previous Message ID                                  |
+-----------------------------------------------------------------+
```

Fig. 4 Reliability Option Format

Current Message ID and Previous Message ID will be added in the packets.

For example, there are two subscriber A and B,

- o Message IDs for the generator are : [1, 2, 3, 4, 5, 6, 7, 8, 9], in which Subscriber A subscribes [1,2,3,6,7] and Subscriber B subscribes [1,2,4,5,7,8,9].

- o Subscriber A will receive [Previous Message ID, Current Message ID] like: [0,1][1,2][2,3][3,6][6,7].

- o Subscriber B will receive [Previous Message ID, Current Message ID] like: [0,1][1,2][2,4][4,5][5,7][7,8][8,9].

5.3.2. Fragmentation Option

UDP palyload has a theoretical length limitation to 65535. Other encapsulation headers will make the actual payload even shorter. Binary encodings like GPB and CBOR can make the message compact. So that the message can be encapsulated within one UDP packet, hence fragmentation will not easily happen. However, text encodings like JSON and XML can easily make the message exceed the UDP length limitation.

The Fragmentation Option can help not Application layer can split the YANG tree into several leaves. Or table into several rows. But the leaf or the row cannot be split any further. Now we consider a very long path. Since the GPB and CBOR are so compact, it’s easy to fit into a UDP packet. But for JSON or XML, it is possible that even one leaf will exceed the UDP boundary.

```
 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
+-----------------------------------------------------------+-+
|            Fagment Number                                 |L|
+-----------------------------------------------------------+-+
```

Fig. 5 Fragmentation Option Format

The Fragmentation Option is available in the message header when the fragmentation flag is set to 1. The option contains:

Fragment Number: indicates the sequence number of the current fragment.

L: is a flag to indicate whether the current fragment is the last one. When 0 is set, current fragment is not the last one, hence more fragments are expected. When 1 is set, current fragment is the last one.

5.4. Data Encoding

Subscribed data can be encoded in GPB, CBOR, XML or JSON format. It is conceivable that additional encodings may be supported as options in the future. This can be accomplished by augmenting the subscription data model with additional identity statements used to refer to requested encodings.

Implementation may support different encoding method per subscription. When bundled notifications is supported between the publisher and the receiver, only subscribed notifications with the same encoding can be bundled as one message.

6. Using DTLS to Secure UPC

The Datagram Transport Layer Security (DTLS) protocol [RFC6347] is designed to meet the requirements of applications that need secure datagram transport.

DTLS can be used as a secure transport to counter all the primary threats to UDP based Publication Channel:

- o Confidentiality to counter disclosure of the message contents.
- o Integrity checking to counter modifications to a message on a hop-by-hop basis.
- o Server or mutual authentication to counter masquerade.

In addition, DTLS also provides:

- o A cookie exchange mechanism during handshake to counter Denial of Service attacks.
- o A sequence number in the header to counter replay attacks.

6.1. Transport

As shown in Figure 6, the DTLS is layered next to the UDP transport is to provide reusable security and authentication functions over UDP. No DTLS extension is required to enable UPC messages over DTLS.

```
+-----------------------------+
|           UPC Message       |
+-----------------------------+
|            DTLS             |
+-----------------------------+
|            UDP              |
+-----------------------------+
|            IP               |
+-----------------------------+
```

Fig. 6: Protocol Stack for DTLS secured UPC

The application implementer will map a unique combination of the remote address, remote port number, local address, and local port number to a session.

Each UPC message is delivered by the DTLS record protocol, which assigns a sequence number to each DTLS record. Although the DTLS implementer may adopt a queue mechanism to resolve reordering, it may not assure that all the messages are delivered in order when mapping on the UDP transport.

Since UDP is an unreliable transport, with DTLS, an originator or relay may not realize that a collector has gone down or lost its DTLS connection state, so messages may be lost.

The DTLS record has its own sequence number, the encryption and decryption will done by DTLS layer, UPC Message layer will not concern this.

6.2. Port Assignment

The Publisher is always a DTLS client, and the Receiver is always a DTLS server. The Receivers MUST support accepting UPC Messages on the UDP port PORT-Y, but MAY be configurable to listen on a different port. The Publisher MUST support sending UPC messages to the UDP port PORT-Y, but MAY be configurable to send messages to a different port. The Publisher MAY use any source UDP port for transmitting messages.

6.3. DTLS Session Initiation

The Publisher initiates a DTLS connection by sending a DTLS Client Hello to the Receiver. Implementations MUST support the denial of service countermeasures defined by DTLS. When these countermeasures are used, the Receiver responds with a DTLS Hello Verify Request containing a cookie. The Publisher responds with a DTLS Client Hello containing the received cookie, which initiates the DTLS handshake.

The Publisher MUST NOT send any UPC messages before the DTLS handshake has successfully completed.

Implementations MUST support DTLS 1.0 [RFC4347] and MUST support the mandatory to implement cipher suite, which is TLS_RSA_WITH_AES_128_CBC_SHA [RFC5246] as specified in DTLS 1.0. If additional cipher suites are supported, then implementations MUST NOT negotiate a cipher suite that employs NULL integrity or authentication algorithms.

Where privacy is REQUIRED, then implementations must either negotiate a cipher suite that employs a non-NULL encryption algorithm or else achieve privacy by other means, such as a physically secured network.

6.4. Sending Data

All UPC messages MUST be sent as DTLS "application data". It is possible that multiple UPC messages be contained in one DTLS record, or that a publication message be transferred in multiple DTLS records. The application data is defined with the following ABNF [RFC5234] expression:

```
APPLICATION-DATA = 1*UPC-FRAME

UPC-FRAME = MSG-LEN SP UPC-MSG

MSG-LEN = NONZERO-DIGIT *DIGIT

SP = %d32

NONZERO-DIGIT = %d49-57

DIGIT = %d48 / NONZERO-DIGIT
```

UPC-MSG is defined in section 5.2.

6.5. Closure

A Publisher MUST close the associated DTLS connection if the connection is not expected to deliver any UPC Messages later. It MUST send a DTLS close_notify alert before closing the connection. A Publisher (DTLS client) MAY choose to not wait for the Receiver’s close_notify alert and simply close the DTLS connection. Once the Receiver gets a close_notify from the Publisher, it MUST reply with a close_notify.

When no data is received from a DTLS connection for a long time (where the application decides what "long" means), Receiver MAY close

the connection. The Receiver (DTLS server) MUST attempt to initiate an exchange of close_notify alerts with the Publisher before closing the connection. Receivers that are unprepared to receive any more data MAY close the connection after sending the close_notify alert.

Although closure alerts are a component of TLS and so of DTLS, they, like all alerts, are not retransmitted by DTLS and so may be lost over an unreliable network.

7. Congestion Control

Congestion control mechanisms that respond to congestion by reducing traffic rates and establish a degree of fairness between flows that share the same path are vital to the stable operation of the Internet [RFC2914]. While efficient, UDP has no build-in congestion control mechanism. Because streaming telemetry can generate unlimited amounts of data, transferring this data over UDP is generally problematic. It is not recommended to use the UDP based publication channel over congestion-sensitive network paths. The only environments where the UDP based publication channel MAY be used are managed networks. The deployments require the network path has been explicitly provisioned for the UDP based publication channel through traffic engineering mechanisms, such as rate limiting or capacity reservations.

8. IANA Considerations

This RFC requests that IANA assigns three UDP port numbers in the "Registered Port Numbers" range with the service names "upc" and "upc-dtls". These ports will be the default ports for the UDP based Publication Channel for NETCONF and RESTCONF. Below is the registration template following the rules in [RFC6335].

Service Name: upc

Transport Protocol(s): UDP

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: UDP based Publication Channel

Reference: RFC XXXX

Port Number: PORT-X

Service Name: upc-dtls

Transport Protocol(s): UDP

Assignee: IESG <iesg@ietf.org>

Contact: IETF Chair <chair@ietf.org>

Description: UDP based Publication Channel (DTLS)

Reference: RFC XXXX

Port Number: PORT-Y

9. Security Considerations

TBD

10. Acknowledgements

The authors of this documents would like to thank Eric Voit, Tim Jenkins, and Huiyang Yang for the initial comments.

11. References

11.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <https://www.rfc-editor.org/info/rfc2914>.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, DOI 10.17487/RFC4347, April 2006, <https://www.rfc-editor.org/info/rfc4347>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://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, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

11.2. Informative References

[I-D.ietf-netconf-netconf-event-notifications]
Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "NETCONF Support for Event Notifications", draft-ietf-netconf-netconf-event-notifications-09 (work in progress), May 2018.

[I-D.ietf-netconf-notification-messages]
Voit, E., Birkholz, H., Bierman, A., Clemm, A., and T. Jenkins, "Notification Message Headers and Bundles", draft-ietf-netconf-notification-messages-03 (work in progress), February 2018.

[I-D.ietf-netconf-restconf-notif]
Voit, E., Rahman, R., Nilsen-Nygaard, E., Clemm, A., and A. Bierman, "RESTCONF and HTTP Transport for Event Notifications", draft-ietf-netconf-restconf-notif-06 (work in progress), June 2018.

[I-D.ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Customized Subscriptions to a Publisher's Event Streams", draft-ietf-netconf-subscribed-notifications-13 (work in progress), June 2018.

[I-D.zhou-netconf-multi-stream-originators]
Zhou, T., Zheng, G., Voit, E., Clemm, A., and A. Bierman, "Subscription to Multiple Stream Originators", draft-zhou-netconf-multi-stream-originators-02 (work in progress), May 2018.

11.3. URIs

[1] https://developers.google.com/protocol-buffers/

Appendix A. Change Log

(To be removed by RFC editor prior to publication)

A.1. draft-ietf-zheng-udp-pub-channel-00 to v00

- o Modified the message header format.
- o Added a section on the Authentication Option.
- o Cleaned up the text and removed unnecessary TBDs.

A.2. v01

- o Removed the detailed description on distributed data collection mechanism from this document. Mainly focused on the description of a UDP based publication channel for telemetry use.
- o Modified the message header format.

A.2. v02

- o Add the section on the transport mechanism.
- o Modified the fixed message header format.

- o Add the fragmentation option for the message header.

A.2. v03

- o Clarify term through the document.
- o Add a section on DTLS support.

Authors' Addresses

Guangying Zheng
Huawei
101 Yu-Hua-Tai Software Road
Nanjing, Jiangsu
China

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

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: January 2, 2019

A. Clemm
Huawei
E. Voit
Cisco Systems
A. Gonzalez Prieto
VMware
A. Tripathy
E. Nilsen-Nygaard
Cisco Systems
A. Bierman
YumaWorks
B. Lengyel
Ericsson
July 1, 2018

YANG Datastore Subscription
draft-ietf-netconf-yang-push-17

Abstract

Via the mechanism described in this document, subscriber applications may request a continuous, customized stream of updates from a YANG datastore. Providing such visibility into changes made upon YANG configuration and operational objects enables new capabilities based on the remote mirroring of configuration and operational state.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 2, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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. Subscription Model 5
3.2. Negotiation of Subscription Policies 6
3.3. On-Change Considerations 7
3.4. Reliability Considerations 8
3.5. Data Encodings 9
3.6. Defining the Selection with a Datastore 10
3.7. Streaming Updates 11
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 17
4.1. Overview 17
4.2. Subscription configuration 22
4.3. YANG Notifications 23

4.4. YANG RPCs 24
5. YANG module 29
6. IANA Considerations 46
7. Security Considerations 46
8. Acknowledgments 47
9. References 48
9.1. Normative References 48
9.2. Informative References 49
Appendix A. Appendix A: Subscription Errors 50
A.1. RPC Failures 50
A.2. Notifications of Failure 51
Appendix B. Changes between revisions 51
Authors' Addresses 55

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 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 Streams" [I-D.draft-ietf-netconf-subscribed-notifications]. Supplementing that work are YANG data model augmentations, extended RPCs, and new

datastore specific update notifications. Transport options for [I-D.draft-ietf-netconf-subscribed-notifications] will work seamlessly with this solution.

2. Definitions and Acronyms

This document uses the terminology defined in [RFC7950], [RFC8341], and [RFC8342]. In addition, the following terms are introduced:

- o Datastore node: An instance of management information in a datastore. Also known as "object".
- o Datastore node update: A data item containing the current value of a datastore node at the time the datastore node update was created.
- o Datastore subscription: A subscription to updates regarding contents of a datastore.
- o Datastore subtree: An instantiated datastore node and the datastore nodes that are hierarchically contained within it.
- o On-change subscription: A datastore subscription with updates that are triggered when changes in subscribed datastore nodes are detected.
- o Periodic subscription: A datastore subscription with updates that are triggered periodically according to some time interval.
- o Selection filter: Evaluation and/or selection criteria, which may be applied against a targeted set of objects.
- o Update record: A representation of one or more datastore node updates. In addition, an update record may contain which type of update led to the datastore node update (e.g., whether the datastore node was added, changed, deleted). Also included in the update record may be other metadata, such as a subscription identifier of the subscription as part of which the update record was generated.
- o Update trigger: A mechanism that determines when an update record needs to be generated.
- o YANG-Push: The subscription and push mechanism for datastore updates that is specified in this document.

3. Solution Overview

This document specifies a solution that provides subscription service for updates from a datastore. This solution supports dynamic as well as configured subscriptions to updates of datastore nodes. Subscriptions specify when 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. Subscription Model

YANG-push subscriptions are defined using a data model that is itself defined in YANG. This model enhances the subscription model defined in [I-D.draft-ietf-netconf-subscribed-notifications] with capabilities that allow subscribers to subscribe to datastore node updates, specifically to specify the update triggers defining 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 datastore nodes and/or subtrees within a datastore for which updates are to be pushed.

- o Specification of update policies contain conditions which trigger the generation and pushing of new update records. There are two types of subscriptions, distinguished by how updates are triggered: periodic and on-change.

  - * For periodic subscriptions, the update 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, an update trigger occurs whenever a change in the subscribed information is detected. Included are additional parameters such as:

    - + Dampening period: In an on-change subscription, detected object changes should be sent as quickly as possible. However it may be undesirable to send a rapid series of object changes. Such behavior has the potential to exhaust of resources in the publisher or receiver. In order to protect against that, a dampening period MAY be used to specify the interval which must pass before successive update records for the same subscription are generated for a receiver. The dampening period collectively applies to the

set of all datastore nodes selected by a single subscription and sent to a single receiver. This means that when there is a change to one or more subscribed objects, an update record containing those objects is created either immediately when no dampening period is in effect, or at the end of a dampening period. If multiple changes to a single object occur during a dampening period, only the value that is in effect at the time the update record is created is included. The dampening period goes into effect every time an update record completes assembly.

+ 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 early synchronization establishes the frame of reference for subsequent updates.

o An encoding (using anydata) for the contents of periodic and on-change push updates.

3.2. Negotiation of Subscription Policies

A dynamic subscription request SHOULD be declined if a publisher’s assessment is that it may be unable to provide update records meeting the terms of an "establish-subscription" or "modify-subscription" rpc request. In this case, a subscriber may quickly follow up with a new rpc 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 subscription supports a simple negotiation between subscribers and publishers for subscription parameters. This negotiation is in the form of supplemental information which may be inserted within error responses to a failed rpc request. This returned error response information, when considered, should increase the likelihood of success for subsequent rpc requests. Such hints include suggested periodic time intervals, acceptable dampening periods, and size estimates for the number or objects which would be returned from a proposed selection filter. However, there are no guarantees that subsequent requests which consider these hints will be accepted.

3.3. On-Change Considerations

On-change subscriptions allow subscribers to receive updates whenever changes to targeted objects occur. As such, on-change subscriptions are particularly effective for data that changes infrequently, yet for which applications need to be quickly 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 publisher implementation. A publisher 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 subscriber to determine whether objects support on-change subscriptions, objects are marked accordingly on a publisher. Accordingly, when subscribing, it is the responsibility of the subscriber 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 publisher 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 MAY be suspended.

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 include 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.

On-change subscriptions can be refined to let users subscribe only to certain types of changes. For example, a subscriber might only want

object creations and deletions, but not modifications of object values.

Putting it all together, following is the conceptual process for creating an push-change-update notification:

1. Just before a change, or at the start of a dampening period, evaluate any filtering and any access control rules. The result is a set "A" of datastore nodes and subtrees.

2. Just after a change, or at the end of a dampening period, evaluate any filtering and any (possibly new) access control rules. The result is a set "B" of datastore nodes and subtrees.

3. Construct a YANG patch record for going from A to B.

4. If there were any changes made between A and B which canceled each other out, insert into the YANG patch record the last change made for any object which otherwise wouldn’t have appeared.

5. If the resulting patch record is non-empty, send it to the receiver.

Note: In cases where a subscriber wants to have separate dampening periods for different objects, multiple subscriptions with different objects in a selection filter can be created.

3.4. Reliability Considerations

A subscription to updates from a datastore is intended to obviate the need for polling. However, in order to do so, it is critical that subscribers can rely on the subscription and have confidence that they will indeed receive the subscribed updates without having to worry about updates being silently dropped. In other words, a subscription constitutes a promise on the side of the publisher to provide the receivers with updates per the terms of the subscription.

Now, there are many reasons why a publisher 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 objects from being collected. For this reason, the solution that is defined in this document mandates that a publisher notifies receivers immediately and reliably whenever it encounters a situation in which it is unable to keep the terms of the subscription, and provides the publisher with the option to suspend the subscription in such a case.

A publisher SHOULD reject a request for a subscription if it is unlikely that the publisher will be able fulfill the terms of that subscription request. In such cases, it is preferable to have a subscriber request a less resource intensive subscription than to deal with frequently degraded behavior.

3.5. Data Encodings

3.5.1. Periodic Subscriptions

In a periodic subscription, the data included as part of an update corresponds to data that could have been read using a retrieval operation.

3.5.2. On-Change Subscriptions

In an on-change subscription, updates need to indicate not only values of changed datastore nodes but also the types of changes that occurred since the last update. Therefore encoding rules for data in on-change updates will generally 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.

A publisher will indicate a change to the effect that a value of a datstore node has been updated by indicating a "replace" operation (applied to the datastore node) in the patch. When a new datastore node was created (other than an element in a list), a publisher will indicate a "create" operation in the patch. When a datastore node was deleted (other than an element in a list), the publisher indicates this by a "delete". When a new list element was created or removed, the publisher indicates it by an "insert" or "remove", respectively.

However a patch must be able to do more than just describe the delta from the previous state to the current state. As per Section 3.3, it must also be able to identify if transient changes have occurred on an object during a dampening period. To support this, it is valid to encode a YANG patch operation so that its application would result in no change between the previous and current state. This indicates that some churn has occurred on the object. An example of this would be a patch that does a "create" operation for a datastore node where the receiver believes one already exists, or a "merge" operation which replaces a previous value with the same value. Note that this means that the "create" and "delete" errors described in [RFC8072] section 2.5 are not errors, and are valid operations with YANG push.

3.6. Defining the Selection with a Datastore

A subscription must specify both the selection filters and the datastore against which these selection filters will be applied. This information is used to choose and subsequently push data from the publisher’s datastore to the receivers.

Only a single selection filter can be applied to a subscription at a time. An rpc request proposing a new selection filter MUST remove any existing filter. The following selection filter types are included in the yang-push data model, and may be applied against a datastore:

- o subtree: A subtree selection filter identifies one or more datastore subtrees. When specified, update records will only come from the datastore nodes of selected datastore subtree(s). The syntax and semantics correspond to that specified for [RFC6241] section 6.

- o xpath: An "xpath" selection filter is an XPath expression that returns a node set. When specified, updates will only come from the selected datastore nodes.

These filters are intended to be used as selectors that define which objects are within the scope of a subscription. A publisher MUST support at least one type of selection filter.

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 misinterpret. Consider an XPath filter which only passes a datastore object when an interface is up. It is up to the receiver to understand implications of the presence or absence of objects in each update.

When the set of selection filtering criteria is applied for a periodic subscription, all selected datastore nodes to which a receiver has access are provided to that receiver. If the same filtering criteria is applied to an on-change subscription, only the subset of those datastore nodes supporting on-change is provided. A datastore node which doesn’t support on-change is never sent as part of an on-change subscription’s "push-update" or "push-change-update".

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 YANG 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 YANG 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" update record contains an instantiated datastore subtree with all of 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 a datastore retrieval operation using the same transport 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 the set of changes that datastore nodes have undergone since the last notification message. In other words, this indicates which datastore 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 "push-update" or "push-change-update" are encoded and placed within notification messages, and ultimately queued for egress over the specified transport.

The following is an example of a notification message for a subscription tracking the operational status of a single Ethernet port (per [RFC8343]). This notification message is encoded XML over NETCONF as per [I-D.draft-ietf-netconf-netconf-event-notifications].

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>2017-10-25T08:00:11.22Z</eventTime>
 <push-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <subscription-id>1011</subscription-id>
   <datastore-contents>
     <interfaces-state xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
       <interface>
         <name>eth0</name>
         <oper-status>up</oper-status>
       </interface>
     </interfaces-state>
   </datastore-contents>
 </push-update>
</notification>
```

Figure 1: Push example

The following is an example of an on-change notification message for the same subscription.

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>2017-10-25T08:22:33.44Z</eventTime>
 <push-change-update xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
   <subscription-id>89</subscription-id>
   <datastore-changes>
     <yang-patch xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-patch">
       <patch-id>1</patch-id>
       <edit>
         <edit-id>edit1</edit-id>
         <operation>merge</operation>
         <target>/ietf-interfaces:interfaces-state</target>
         <value>
           <interfaces-state xmlns="http://foo.com/ietf-interfaces">
             <interface>
               <name>eth0</name>
               <oper-status>down</oper-status>
             </interface>
           </interfaces-state>
         </value>
       </edit>
     </yang-patch>
   </datastore-changes>
 </push-change-update>
</notification>
```

Figure 2: Push example for on change

Of note in the above example is the 'patch-id' with a value of '1'. Per [RFC8072], the 'patch-id' is an arbitrary string. With YANG Push, the publisher SHOULD put into the 'patch-id' a counter starting at '1' which increments with every 'push-change-update' generated for a subscription. If used as a counter, this counter MUST be reset to '1' anytime a resynchronization occurs (i.e., with the sending of a 'push-update'). Also if used as a counter, the counter MUST be reset to '1' the after passing a maximum value of '4294967295' (i.e. maximum value that can be represented using uint32 data type). Such a mechanism allows easy identification of lost or out-of-sequence update records.

3.8. Subscription Management

The RPCs defined within [I-D.draft-ietf-netconf-subscribed-notifications] have been enhanced to support datastore subscription negotiation. Also, new error codes have been added that are able to indicate why a datastore subscription attempt has failed, along with new yang-data that MAY be used to include details on input parameters that might result in a successful subsequent RPC invocation.

The establishment or modification of a datastore subscription can be rejected for multiple reasons. This includes a too large subtree request, or the inability of the publisher to 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.

In the case of a rejected request for an establishment of a datastore subscription, the hints MUST be transported within a yang-data "establish-subscription-datastore-error-info" container inserted into the RPC error response, in lieu of the "establish-subscription-stream-error-info" that is inserted in case of a stream subscription.

Below is a tree diagram for "establish-subscription-datastore-error-info". All tree diagrams used in this document follow the notation defined in [RFC8340]

```
yang-data establish-subscription-datastore-error-info
   +--ro establish-subscription-datasore-error-info
      +--ro reason?                identityref
      +--ro period-hint?           yang:timeticks
      +--ro filter-failure-hint?   string
      +--ro object-count-estimate? uint32
      +--ro object-count-limit?    uint32
      +--ro kilobytes-estimate?    uint32
      +--ro kilobytes-limit?       uint32
```

Figure 3: Tree diagram for establish-subscription-datastore-error-info

Similarly, in the case of a rejected request for modification of a datastore subscription, the hints MUST be transported within a yang-data "modify-subscription-datastore-error-info" container inserted into the RPC error response, in lieu of the "modify-subscription-stream-error-info" that is inserted in case of a stream subscription.

Below is a tree diagram for "modify-subscription-datastore-error-info".

```
yang-data modify-subscription-datastore-error-info
   +--ro modify-subscription-datasore-error-info
      +--ro reason?                identityref
      +--ro period-hint?           yang:timeticks
      +--ro filter-failure-hint?   string
      +--ro object-count-estimate? uint32
      +--ro object-count-limit?    uint32
      +--ro kilobytes-estimate?    uint32
      +--ro kilobytes-limit?       uint32
```

Figure 4: Tree diagram for modify-subscription-datastore-error-info

3.9. Receiver Authorization

A receiver of subscription data MUST only be sent updates for which they have proper authorization. A publisher 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 datastore subtrees. This enables YANG data pushed based on subscriptions to be authorized equivalently to a regular data retrieval (get) operation.

A publisher MUST allow for the possibility that a subscription’s selection filter references non-existent or access-protected data. Such support permits a receiver the ability to monitor the entire

lifecyle of some datastore tree. In this case, all "push-update" notifications must be sent empty, and no "push-change-update" notifications will be sent until some data becomes visible for a receiver.

A publisher MAY choose reject an establish-subscription request which selects non-existent or access-protected data. In addition, a publisher MAY choose to terminate a dynamic subscription or suspend a configured receiver when the authorization privileges of a receiver change, or the access controls for subscribed objects change. Such a capability enables the publisher to avoid having to support a continuous, and total filtering of an entire subscription’s content.

In these cases above, the error identity "unchanging-selection" SHOULD be returned. This reduces the possibility of leakage of access controlled objects.

Each "push-update" and "push-change-update" MUST have access control applied. This includes validating that read access is permitted for any new objects selected since the last notification message was sent to a particular each receiver. To accomplish this, implementations SHOULD support the conceptual authorization model of [RFC8342], specifically section 3.2.4.

```
                      +-----------------+      +--------------------+
  push-update or -->  | datastore node  |  yes | add datastore node |
 push-change-update   | access allowed? | ---> | to update message  |
                      +-----------------+      +--------------------+
```

Figure 5: Updated [rfc6536bis] access control for push updates

If read access into previously accessible nodes has been lost due to a receiver permissions change, this SHOULD be reported as a patch "delete" operation for on-change subscriptions. If not capable of handling such receiver permission changes with such a "delete", publisher implementations MUST force dynamic subscription re-establishment or configured subscription re-initialization so that appropriate filtering is installed.

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 datastore node changes frequently (e.g., [RFC8343]’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.

In those cases, it will be important for client applications to have a way to identify for which objects on-change notifications are supported and for which ones they are not supported. Otherwise client applications will have no way of knowing whether they can indeed rely on their on-change subscription to provide them with the change updates that they are interested in. In other words, if implementations do not provide a solution and do not support comprehensive on-change notifiability, clients of those implementations will have no way of knowing what their on-change subscription actually covers.

Implementations are therefore strongly advised to provide a solution to this problem. It is expected that such a solution will be standardized at some point in the future. In the meantime and until this occurs, implementations will be expected to provide their own solution.

3.11. Other Considerations

3.11.1. Robustness and reliability

Particularly in the case of on-change updates, it is important that these updates do not get lost. Or in case the loss of an update is unavoidable, it is critical that the receiver is notified accordingly.

Update records for a single subscription MUST NOT be resequenced prior to transport.

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 "incomplete-update" flag on any update record which is known to be missing information.

- o It MAY choose to suspend a subscription as per [I-D.draft-ietf-netconf-subscribed-notifications].

- 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 "no-synch-on-start" option is not present for the subscription, then the full datastore contents MAY be sent via a "push-update" instead (effectively replacing the previous contents). If neither of these are possible, then an "incomplete-update" flag MUST be included on the next "push-change-update".

Note: It is perfectly acceptable to have a series of "push-change-update" notifications (and even "push update" notifications) 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. 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 update trigger (on-change or periodic), the period in which to report changes (one second periods will consume more resources than one hour periods), the amount of data in the datastore 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. The tree diagram follows the notiation defined in [RFC8340]. New schema objects defined here (i.e., beyond those from [I-D.draft-ietf-netconf-subscribed-notifications]) are identified with "yp". For the reader’s convenience, in order to compact the tree representation, some nodes that are defined in ietf-subscribed-notifications and that are not essential to the understanding of the data model defined here have been removed. This is indicated by "..." in the diagram where applicable.

```
module: ietf-subscribed-notifications
    ...
    +--rw filters
    |  ...
    |  +--rw yp:selection-filter* [identifier]
    |     +--rw yp:identifier                 sn:filter-id
    |     +--rw (yp:filter-spec)?
    |        +--:(yp:datastore-subtree-filter)
    |        |  +--rw yp:datastore-subtree-filter? <anydata>
    |        |          {sn:subtree}?
    |        +--:(yp:datastore-xpath-filter)
    |           +--rw yp:datastore-xpath-filter?   yang:xpath1.0
    |                   {sn:xpath}?
    +--rw subscriptions
       +--rw subscription* [identifier]
```

```
             |  ...
             +--rw (target)
             |  +--:(stream)
             |  |   ...
             |  +--:(yp:datastore)
             |     +--rw yp:datastore             identityref
             |     +--rw (yp:selection-filter)?
             |        +--:(yp:by-reference)
             |        |  +--rw yp:selection-filter-ref
             |        |          selection-filter-ref
             |        +--:(yp:within-subscription)
             |           +--rw (yp:filter-spec)?
             |              +--:(yp:datastore-subtree-filter)
             |              |  +--rw yp:datastore-subtree-filter?
             |              |          <anydata> {sn:subtree}?
             |              +--:(yp:datastore-xpath-filter)
             |                 +--rw yp:datastore-xpath-filter?
             |                         yang:xpath1.0 {sn:xpath}?
             | ...
             +--rw (yp:update-trigger)?
                +--:(yp:periodic)
                |  +--rw yp:periodic!
                |     +--rw yp:period         yang:timeticks
                |     +--rw yp:anchor-time?   yang:date-and-time
                +--:(yp:on-change) {on-change}?
                   +--rw yp:on-change!
                      +--rw yp:dampening-period?    yang:timeticks
                      +--rw yp:no-synch-on-start?   empty
                      +--rw yp:excluded-change*     change-type

     rpcs:
       +---x establish-subscription
       |  +---w input
       |  |  ...
       |  |  +---w (target)
       |  |  |  +--:(stream)
       |  |  |  |  ...
       |  |  |  +--:(yp:datastore)
       |  |  |     +---w yp:datastore                   identityref
       |  |  |     +---w (yp:selection-filter)?
       |  |  |        +--:(yp:by-reference)
       |  |  |        |  +---w yp:selection-filter-ref
       |  |  |        |          selection-filter-ref
       |  |  |        +--:(yp:within-subscription)
       |  |  |           +---w (yp:filter-spec)?
       |  |  |              +--:(yp:datastore-subtree-filter)
       |  |  |              |  +---w yp:datastore-subtree-filter?
       |  |  |              |          <anydata> {sn:subtree}?
```

```
       |  |  |             +--:(yp:datastore-xpath-filter)
       |  |  |                +---w yp:datastore-xpath-filter?
       |  |  |                        yang:xpath1.0 {sn:xpath}?
       |  |  | ...
       |  |  +---w (yp:update-trigger)?
       |  |     +--:(yp:periodic)
       |  |     |  +---w yp:periodic!
       |  |     |     +---w yp:period         yang:timeticks
       |  |     |     +---w yp:anchor-time?   yang:date-and-time
       |  |     +--:(yp:on-change) {on-change}?
       |  |        +---w yp:on-change!
       |  |           +---w yp:dampening-period?    yang:timeticks
       |  |           +---w yp:no-synch-on-start?   empty
       |  |           +---w yp:excluded-change*     change-type
       |  +--ro output
       |     +--ro identifier    subscription-id
       +---x modify-subscription
       |  +---w input
       |     ...
       |     +---w (target)
       |     |  ...
       |     |  +--:(yp:datastore)
       |     |     +---w (yp:selection-filter)?
       |     |        +--:(yp:by-reference)
       |     |        |  +---w yp:selection-filter-ref
       |     |        |          selection-filter-ref
       |     |        +--:(yp:within-subscription)
       |     |           +---w (yp:filter-spec)?
       |     |              +--:(yp:datastore-subtree-filter)
       |     |              |  +---w yp:datastore-subtree-filter?
       |     |              |          <anydata> {sn:subtree}?
       |     |              +--:(yp:datastore-xpath-filter)
       |     |                 +---w yp:datastore-xpath-filter?
       |     |                         yang:xpath1.0 {sn:xpath}?
       |     | ...
       |     +---w (yp:update-trigger)?
       |        +--:(yp:periodic)
       |        |  +---w yp:periodic!
       |        |     +---w yp:period         yang:timeticks
       |        |     +---w yp:anchor-time?   yang:date-and-time
       |        +--:(yp:on-change) {on-change}?
       |           +---w yp:on-change!
       |              +---w yp:dampening-period?   yang:timeticks
       +---x delete-subscription
       |  ...
       +---x kill-subscription
          ...
```

```
     yang-data (for placement into rpc error responses)
       ...

     notifications:
       +---n replay-completed {replay}?
       |  ...
       +---n subscription-completed
       |  ...
       +---n subscription-started {configured}?
       |  |  ...
       |  +--ro (target)
       |  |  ...
       |  |  +--:(yp:datastore)
       |  |     +--ro yp:datastore                   identityref
       |  |     +--ro (yp:selection-filter)?
       |  |        +--:(yp:by-reference)
       |  |        |  +--ro yp:selection-filter-ref
       |  |        |          selection-filter-ref
       |  |        +--:(yp:within-subscription)
       |  |           +--ro (yp:filter-spec)?
       |  |              +--:(yp:datastore-subtree-filter)
       |  |              |  +--ro yp:datastore-subtree-filter?
       |  |                         <anydata> {sn:subtree}?
       |  |              +--:(yp:datastore-xpath-filter)
       |  |                 +--ro yp:datastore-xpath-filter?
       |  |                         yang:xpath1.0 {sn:xpath}?
       |  ...
       |  +--ro (yp:update-trigger)?
       |     +--:(yp:periodic)
       |     |  +--ro yp:periodic!
       |     |     +--ro yp:period         yang:timeticks
       |     |     +--ro yp:anchor-time?   yang:date-and-time
       |     +--:(yp:on-change) {on-change}?
       |        +--ro yp:on-change!
       |           +--ro yp:dampening-period?    yang:timeticks
       |           +--ro yp:no-synch-on-start?   empty
       |           +--ro yp:excluded-change*     change-type
       +---n subscription-resumed
       |  ...
       +---n subscription-modified
       |  ...
       |  +--ro (target)
       |  |  |  ...
       |  |  +--:(yp:datastore)
       |  |     +--ro yp:datastore                   identityref
       |  |     +--ro (yp:selection-filter)?
       |  |        +--:(yp:by-reference)
       |  |        |  +--ro yp:selection-filter-ref
```

```
       |  |        |          selection-filter-ref
       |  |        +--:(yp:within-subscription)
       |  |           +--ro (yp:filter-spec)?
       |  |              +--:(yp:datastore-subtree-filter)
       |  |              |  +--ro yp:datastore-subtree-filter?
       |  |              |          <anydata> {sn:subtree}?
       |  |              +--:(yp:datastore-xpath-filter)
       |  |                 +--ro yp:datastore-xpath-filter?
       |  |                         yang:xpath1.0 {sn:xpath}?
       |  ...
       |  +--ro (yp:update-trigger)?
       |     +--:(yp:periodic)
       |     |  +--ro yp:periodic!
       |     |     +--ro yp:period         yang:timeticks
       |     |     +--ro yp:anchor-time?   yang:date-and-time
       |     +--:(yp:on-change) {on-change}?
       |        +--ro yp:on-change!
       |           +--ro yp:dampening-period?    yang:timeticks
       |           +--ro yp:no-synch-on-start?   empty
       |           +--ro yp:excluded-change*     change-type
       +---n subscription-terminated
       |  ...
       +---n subscription-suspended
          ...

   module: ietf-yang-push

     rpcs:
       +---x resynch-subscription {on-change}?
          +---w input
             +---w identifier    sn:subscription-id

     yang-data: (for placement into rpc error responses)
       +-- resynch-subscription-error
       |  +--ro reason?                   identityref
       |  +--ro period-hint?              timeticks
       |  +--ro filter-failure-hint?      string
       |  +--ro object-count-estimate?    uint32
       |  +--ro object-count-limit?       uint32
       |  +--ro kilobytes-estimate?       uint32
       |  +--ro kilobytes-limit?          uint32
       +-- establish-subscription-error-datastore
       |  +--ro reason?                   identityref
       |  +--ro period-hint?              timeticks
       |  +--ro filter-failure-hint?      string
       |  +--ro object-count-estimate?    uint32
       |  +--ro object-count-limit?       uint32
       |  +--ro kilobytes-estimate?       uint32
```

```
       |  +--ro kilobytes-limit?          uint32
       +-- modify-subscription-error-datastore
          +--ro reason?                   identityref
          +--ro period-hint?              timeticks
          +--ro filter-failure-hint?      string
          +--ro object-count-estimate?    uint32
          +--ro object-count-limit?       uint32
          +--ro kilobytes-estimate?       uint32
          +--ro kilobytes-limit?          uint32

     notifications:
       +---n push-update
       |  +--ro subscription-id?      sn:subscription-id
       |  +--ro incomplete-update?     empty
       |  +--ro datastore-contents?   <anydata>
       +---n push-change-update {on-change}?
          +--ro subscription-id?     sn:subscription-id
          +--ro incomplete-update?    empty
          +--ro datastore-changes?   <anydata>
```

Figure 6: 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". New parameters extending the basic subscription data model in [I-D.draft-ietf-netconf-subscribed-notifications] include:

- o The targeted datastore from which the selection is being made. The potential datastores include those from [RFC8341]. A platform may also choose to support a custom datastore.

- o A selection filter identifying yang nodes of interest within a datastore. Filter contents are specified via a reference to an existing filter, or via an in-line definition for only that subscription. Referenced filters allows 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 can be calculated from the periodic parameters:

* a "period" which defines the duration between push updates.

* an "anchor-time"; update intervals always fall on the points in time that are a multiple of a "period" from an "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 any dampening period has completed, triggering 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.

4.3. YANG Notifications

4.3.1. State Change Notifications

Subscription state notifications and mechanism are reused from [I-D.draft-ietf-netconf-subscribed-notifications]. Notifications "subscription-started" and "subscription-modified" have been augmented to include the datastore specific objects.

4.3.2. Notifications for Subscribed Content

Along with the subscribed content, there are other objects which might be part of a "push-update" or "push-change-update" notification.

A "subscription-id" MUST be transported along with the subscribed contents. An [RFC5277] Section 4 one-way notification MAY be used for encoding updates. Where it is, the relevant "subscription-id" MUST be encoded as the first element within each "push-update" or "push-change-update". This allows a receiver to differentiate which subscription resulted in a particular push.

A "time-of-update" which represents the time an update record snapshot was generated. A receiver MAY assume that a publisher's objects have these pushed values at this point in time.

An "incomplete-update" leaf. This leaf 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 was unable to providing the full set of datastore nodes to a publisher process.) To facilitate re-synchronization of on-change subscriptions, a publisher MAY subsequently send a "push-update" containing a full selection snapshot of subscribed data.

4.4. YANG RPCs

YANG-Push subscriptions are established, modified, and deleted using RPCs augmented from [I-D.draft-ietf-netconf-subscribed-notifications].

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-subscribed-notifications"
       xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore>
      <yp:source xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
        ds:operational
      </yp:source>
      <xpath-filter
          xmlns:ex="http://example.com/sample-data/1.0"
          select="/ex:foo"/>
    </yp:datastore>
    <yp:periodic>
      <yp:period>500</yp:period>
    </yp:periodic>
 </establish-subscription>
</netconf:rpc>
```

Figure 7: Establish-subscription RPC

A positive response includes the "identifier" 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-subscribed-notifications">
       ok
    </subscription-result>
    <identifier
      xmlns="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
       52
    </identifier>
</rpc-reply>
```

Figure 8: Establish-subscription positive RPC response

A subscription can be rejected for multiple reasons, including the lack of authorization to establish a subscription, no capacity to serve the subscription at the publisher, or the inability of the publisher to select datastore content at the requested cadence.

If a request is rejected because the publisher is not able to serve it, the publisher SHOULD include in the returned error hints which help a subscriber understand subscription parameters might have been accepted for the request. These hints would be included within the yang-data structure "establish-subscription-error-datastore". However even with these hints, there are no guarantee that subsequent requests will in fact be accepted.

The specific parameters to be returned in as part of the RPC error response depend on the specific transport that is used to manage the subscription. In the case of NETCONF [I-D.draft-ietf-netconf-netconf-event-notifications], when a subscription request is rejected, the NETCONF RPC reply MUST include an "rpc-error" element with the following elements:

- o "error-type" of "application".
- o "error-tag" of "operation-failed".
- o Optionally, an "error-severity" of "error" (this MAY but does not have to be included).
- o "error-app-tag" with the value being a string that corresponds to an identity with a base of "establish-subscription-error".
- o Optionally, "error-info" containing XML-encoded data with hints for parameter settings that might result in future RPC success per yang-data definition "establish-subscription-error-datastore".

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-subscribed-notifications"
  xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <yp:datastore
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter netconf:type="xpath"
    xmlns:ex="http://example.com/sample-data/1.0">
      /ex:foo
    </yp:datastore-xpath-filter>
    <yp:on-change>
      <yp:dampening-period>100</yp:dampening-period>
    </yp:on-change>
  </establish-subscription>
</netconf:rpc>
```

Figure 9: 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">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>operation-failed</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>
        on-change-unsupported
    </error-message>
    <error-path
   xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
      /yp:periodic/yp:period
    </error-path>
  </rpc-error>
</rpc-reply>
```

Figure 10: 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-subscribed-notifications"
   xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
    <identifier>1011</identifier>
    <yp:datastore
    xmlns:ds="urn:ietf:params:xml:ns:yang:ietf-datastores">
      ds:operational
    </yp:datastore>
    <yp:datastore-xpath-filter
    netconf:type="xpath" xmlns:ex="http://example.com/sample-data/1.0">
      /ex:bar
    </yp:datastore-xpath-filter>
    <yp:periodic>
      <yp:period>250</yp:period>
    </yp:periodic>
   </modify-subscription>
</netconf:rpc>
```

Figure 11: Modify subscription request

The publisher MUST respond explicitly positively or negatively to the request. If the subscription modification is rejected, the subscription is maintained as it was before the modification request. In addition, the publisher MUST send an rpc error response. This rpc error response may contain hints encapsulated within the yang-data structure "modify-subscription-error-datastore". A subscription MAY be modified multiple times.

The specific parameters to be returned in as part of the RPC error response depend on the specific transport that is used to manage the subscription. In the case of NETCONF [I-D.draft-ietf-netconf-netconf-event-notifications], when a subscription request is rejected, the NETCONF RPC reply MUST include an "rpc-error" element with the following elements:

- "error-type" of "application".

- "error-tag" of "operation-failed".

- o Optionally, an "error-severity" of "error" (this MAY but does not have to be included).

- o "error-app-tag" with the value being a string that corresponds to an identity with a base of "modify-subscription-error".

- o "error-path" pointing to the object or parameter that caused the rejection.

- o Optionally, "error-info" containing XML-encoded data with hints for parameter settings that might result in future RPC success per yang-data definition "modify-subscription-error-datastore".

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's "identifier". This RPC is unmodified from [I-D.draft-ietf-netconf-subscribed-notifications].

4.4.4. Resynch-subscription RPC

This RPC is only applicable only for on-change subscriptions previously established using an "establish-subscription" RPC. For example:

```
<netconf:rpc message-id="103"
xmlns:netconf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <resynch-subscription
  xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push"
  xmlns:sn="urn:ietf:params:xml:ns:yang:ietf-subscribed-notifications">
    <identifier>1011</identifier>
  </resynch-subscription>
</netconf:rpc>
```

Resynch subscription

On receipt, a publisher must either accept the request and quickly follow with a "push-update", or send an appropriate error within an rpc error response. Within an error response, the publisher may include supplemental information about the reasons within the yang-data structure "resynch-subscription-error".

4.4.5. 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]. This YANG library information is important for the receiver to reproduce the set of object definitions used within the publisher.

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 publisher implementation). In this case, the receiver needs to be informed of module changes before datastore nodes from changed modules can be processed correctly. The YANG library provides a simple "yang-library-change" notification that informs the subscriber that the library has changed. The receiver then needs to re-read the entire YANG library data for the replicated publisher 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@2018-07-01.yang"
module ietf-yang-push {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-push";
  prefix yp;

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

```
        reference
          "draft-ietf-netconf-subscribed-notifications:
           Customized Subscriptions to a Publisher's Event Streams

           NOTE TO RFC Editor: Please replace above reference to
           draft-ietf-netconf-subscribed-notifications with RFC number
           when published (i.e. RFC xxxx).";
  }
  import ietf-datastores {
    prefix ds;
        reference
          "RFC 8342: Network Management Datastore Architecture (NMDA)";
  }
  import ietf-restconf   {
    prefix rc;
        reference
      "RFC 8040: RESTCONF Protocol";
  }

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

     Editor:   Alexander Clemm
               <mailto:ludwig@clemm.org>

     Editor:   Eric Voit
               <mailto:evoit@cisco.com>

     Editor:   Alberto Gonzalez Prieto
               <mailto:agonzalezpri@vmware.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 YANG specifications for YANG push.
```

```
        Copyright (c) 2018 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
    (https://trustee.ietf.org/license-info).

        This version of this YANG module is part of
        draft-ietf-netconf-yang-push-17; see the RFC itself for full
        legal notices.

        NOTE TO RFC EDITOR: Please replace above reference to
        draft-ietf-netconf-yang-push-17 with RFC number when published
        (i.e. RFC xxxx).";

  revision 2018-07-01 {
    description
      "Initial revision.
          NOTE TO RFC EDITOR:
          (1)Please replace the above revision date to
          the date of RFC publication when published.
          (2) Please replace the date in the file name
          (ietf-yang-push@2018-07-01.yang) to the date of RFC publication.
          (3) Please replace the following reference to
          draft-ietf-netconf-yang-push-17 with RFC number when published
          (i.e. RFC xxxx).";
    reference
      "draft-ietf-netconf-yang-push-17";
  }

 /*
  * FEATURES
  */

  feature on-change {
    description
      "This feature indicates that on-change triggered subscriptions
       are supported.";
  }

 /*
  * IDENTITIES
  */

  /* Error type identities for datastore subscription  */
```

```
  identity resynch-subscription-error {
     description
      "Problem found while attempting to fulfill an
      'resynch-subscription' RPC request. ";
  }

  identity cant-exclude {
    base sn:establish-subscription-error;
    description
      "Unable to remove the set of 'excluded-changes'.  This means the
      publisher is unable to restrict 'push-change-update's to just the
      change types requested for this subscription.";
  }

  identity datastore-not-subscribable {
    base sn:establish-subscription-error;
    base sn:subscription-terminated-reason;
    description
      "This is not a subscribable datastore.";
  }

  identity no-such-subscription-resynch {
    base resynch-subscription-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 configured subscription.";
  }

  identity on-change-unsupported {
    base sn:establish-subscription-error;
    description
      "On-change is not supported for any objects which are selectable
      by this filter.";
  }

  identity on-change-synch-unsupported {
    base sn:establish-subscription-error;
    description
      "Neither synch on start nor resynchronization are supported for
      this subscription.  This error will be used for two reasons.
      First if an 'establish-subscription' RPC doesn't include
      'no-synch-on-start', yet the publisher can't support sending a
      'push-update' for this subscription for reasons other than
      'on-change-unsupported' or 'synchronization-size'.  And second,
      if the 'resynch-subscription' RPC is invoked either for an
      existing periodic subscription, or for an on-change subscription
      which can't support resynchronization.";
```

```
  }

  identity period-unsupported {
    base sn:establish-subscription-error;
    base sn:modify-subscription-error;
    base sn:subscription-suspended-reason;
    description
      "Requested time period is too short. This can be for both
      periodic and on-change subscriptions (with or without
      dampening.) Hints suggesting alternative periods may be
      returned as supplemental information.";
  }

  identity result-too-big {
    base sn:establish-subscription-error;
    base sn:modify-subscription-error;
    base sn:subscription-suspended-reason;
    description
      "Periodic or on-change push update datatrees exceed a maximum
      size limit.  Hints on estimated size of what was too big may
      be returned as supplemental information.";
  }

  identity synchronization-size {
    base sn:establish-subscription-error;
    base sn:modify-subscription-error;
    base resynch-subscription-error;
    base sn:subscription-suspended-reason;
    description
      "Synch-on-start or resynchronization datatree exceeds a maximum
      size limit.  Hints on estimated size of what was too big may be
      returned as supplemental information.";
  }

  identity unchanging-selection {
    base sn:establish-subscription-error;
    base sn:modify-subscription-error;
    base sn:subscription-terminated-reason;
    description
      "Selection filter is unlikely to ever select datatree nodes.  This
      means that based on the subscriber’s current access rights, the
      publisher recognizes that the selection filter is unlikely to ever
      select datatree nodes which change. Examples for this might be
      that node or subtree doesn’t exist, read access is not permitted
      for a receiver, or static objects that only change at reboot have
      been chosen.";
  }
```

```
/*
 * TYPE DEFINITIONS
 */

typedef change-type {
  type enumeration {
    enum "create" {
      description
        "A change that refers to the creation of a new data node.";
    }
    enum "delete" {
      description
        "A change that refers to the deletion of a data node.";
    }
    enum "insert" {
      description
        "A change that refers to the insertion of a new
        user-ordered data node.";
    }
    enum "merge" {
      description
        "A change that refers to a merging of a new value with a
        target data node.";
    }
    enum "move" {
      description
        "A change that refers to a reordering of the target data
        node";
    }
    enum "replace" {
      description
        "A change that refers to a replacement of the target data
        node's value.";
    }
    enum "remove" {
      description
        "A change that refers to the removal of a data node.";
    }
  }
  description
    "Specifies different types of datastore changes.";
  reference
    "RFC 8072 section 2.5, with a delta that it is valid for a
    receiver to process an update record which performs a create
    operation on a datastore node the receiver believes exists, or to
    process a delete on a datastore node the receiver believes is
    missing.";
}
```

```
  typedef selection-filter-ref {
    type leafref {
      path "/sn:filters/yp:selection-filter/yp:filter-id";
    }
    description
      "This type is used to reference a selection filter.";
  }

  /*
   * GROUP DEFINITIONS
   */

  grouping datastore-criteria {
    description
      "A grouping to define criteria for which selected objects from
       a targeted datastore should be included in push updates.";
    leaf datastore {
       type identityref {
         base ds:datastore;
       }
       mandatory true;
       description
         "Datastore from which to retrieve data.";
    }
    uses selection-filter-objects;
  }

  grouping selection-filter-types {
    description
      "This grouping defines the types of selectors for objects from a
       datastore.";
    choice filter-spec {
      description
        "The content filter specification for this request.";
      anydata datastore-subtree-filter {
        if-feature "sn:subtree";
        description
          "This parameter identifies the portions of the
           target datastore to retrieve.";
      }
      leaf datastore-xpath-filter {
        if-feature "sn:xpath";
        type yang:xpath1.0;
        description
          "This parameter contains an XPath expression identifying the
          portions of the target datastore to retrieve.

          If the expression returns a node-set, all nodes in the
```

```
          node-set are selected by the filter.  Otherwise, if the
          expression does not return a node-set, the filter
          doesn't select any nodes.

          The expression is evaluated in the following XPath context:

           o  The set of namespace declarations are those in scope on
              the 'datastore-xpath-filter' leaf element.

           o  The set of variable bindings is empty.

           o  The function library is the core function library, and
              the XPath functions defined in section 10 in RFC 7950.

           o  The context node is the root node of the target
              datastore.";
      }
    }
  }

  grouping selection-filter-objects {
    description
      "This grouping defines a selector for objects from a
       datastore.";
    choice selection-filter {
      description
        "The source of the selection filter applied to the subscription.
        This 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 selection-filter-ref {
          type selection-filter-ref;
          mandatory true;
          description
            "References an existing selection filter which is to be
             applied to the subscription.";
        }
      }
      case within-subscription {
        description
          "Local definition allows a filter to have the same lifecycle
          as the subscription.";
        uses selection-filter-types;

      }
    }
```

```
  }

  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 record to
         the subscriber.";
      case periodic {
        container periodic {
          presence "indicates an periodic subscription";
          description
            "The publisher is requested to notify periodically the
            current values of the datastore as defined by the selection
            filter.";
          leaf period {
            type yang:timeticks;
            mandatory true;
            description
              "Duration of time which should occur between periodic
               push updates.";
          }
          leaf anchor-time {
            type yang:date-and-time;
            description
              "Designates a timestamp before or after which a series of
               periodic push updates are determined. The next update
               will take place at a whole multiple interval from the
               anchor time.  For example, for an anchor time is set for
               the top of a particular minute and a period interval of a
               minute, updates will be sent at the top of every minute
               this subscription is active.";
          }
        }
      }
      case on-change {
        if-feature "on-change";
        container on-change {
          presence "indicates an on-change subscription";
        description
            "The publisher is requested to notify changes in values in
             the datastore subset as defined by a selection filter.";
          leaf dampening-period {
            type yang:timeticks;
            default 0;
```

```
            description
              "Specifies the minimum interval between the assembly of
                successive update records for a single receiver of a
              subscription. Whenever subscribed objects change, and a
              dampening period interval (which may be zero) has elapsed
              since the previous update record creation for a receiver,
              then any subscribed objects and properties which have
              changed since the previous update record will have their
              current values marshalled and placed into a new update
              record.";
          }
        }
      }
    }
  }

  grouping update-policy {
    description
      "This grouping describes the datastore specific subscription
       conditions of a subscription.";
    uses update-policy-modifiable {
      augment "update-trigger/on-change/on-change" {
        description
          "Includes objects not modifiable once subscription is
           established.";
        leaf no-synch-on-start {
          type empty;
          description
            "The presence of this object restricts an on-change
             subscription from sending push-update notifications.  When
             present, pushing a full selection per the terms of the
             selection filter MUST NOT be done for this subscription.
             Only updates about changes, 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.  After that, push-change-update
             notifications are exclusively sent unless the publisher
             chooses to resynch the subscription via a new push-update
             notification.";
        }
        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 hints {
    description
      "Parameters associated with some error for a subscription made
      upon a datastore.";
    leaf period-hint {
      type yang:timeticks;
      description
        "Returned when the requested time period is too short. This
         hint can assert a viable period for either a periodic push
         cadence or an on-change dampening interval.";
    }
    leaf filter-failure-hint {
      type string;
        description
          "Information describing where and/or why a provided filter
           was unsupportable for a subscription.";
    }
    leaf object-count-estimate {
      type uint32;
      description
        "If there are too many objects which could potentially be
         returned by the selection 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
         selection 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 selection 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.";
    }
  }


  /*
   * RPCs
   */

   rpc resynch-subscription {
    if-feature "on-change";
    description
      "This RPC allows a subscriber of an active on-change
       subscription to request a full push of objects.
       A successful invocation results in a push-update of all
       datastore objects that the subscriber is permitted to access.
       This RPC can only be invoked on the same session on which the
       subscription was established (using an establish-subscription
       RPC).  In case of an error, a resynch-subscription-error is
       sent as part of an error response.";
    input {
      leaf identifier {
        type sn:subscription-id;
        mandatory true;
        description
          "Identifier of the subscription that is to be resynched.";
      }
    }
  }

  rc:yang-data resynch-subscription-error {
    container resynch-subscription-error {
      description
        "If a ’resynch-subscription’ RPC fails, the subscription is not
        resynched and the RPC error response MUST indicate the reason
        for this failure. This yang-data MAY be inserted as structured
        data within a subscription’s RPC error response to indicate the
        failure reason.";
      leaf reason {
        type identityref {
          base resynch-subscription-error;
        }
        mandatory true;
        description
          "Indicates the reason why the publisher has declined a request
          for subscription resynchronization.";
      }
      uses hints;
```

```
    }
  }

  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;
  }

  augment "/sn:establish-subscription/sn:input/sn:target" {
    description
      "This augmentation adds the datastore as a valid target
      for the subscription to RPC input.";
    case datastore {
      description
        "Information specifying the parameters of an request for a
        datastore subscription.";
      uses datastore-criteria;
    }
  }

  rc:yang-data establish-subscription-datastore-error-info {
    container establish-subscription-datastore-error-info {
      description
        "If any 'establish-subscription' RPC parameters are
        unsupportable against the datastore, a subscription is not
        created and the RPC error response MUST indicate the reason why
        the subscription failed to be created. This yang-data MAY be
        inserted as structured data within a subscription's RPC error
        response to indicate the failure reason.  This yang-data MUST be
        inserted if hints are to be provided back to the subscriber.";
      leaf reason {
        type identityref {
          base sn:establish-subscription-error;
        }
        description
          "Indicates the reason why the subscription has failed to
          be created to a targeted datastore.";
      }
      uses 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:input/sn:target" {
    description
      "This augmentation adds the datastore as a valid target
      for the subscription to RPC input.";
    case datastore {
      description
        "Information specifying the parameters of an request for a
         datastore subscription.";
      uses selection-filter-objects;
    }
  }

  rc:yang-data modify-subscription-datastore-error-info {
    container modify-subscription-datastore-error-info {
      description
        "This yang-data MAY be provided as part of a subscription's RPC
        error response when there is a failure of a
        'modify-subscription' RPC which has been made against a
        datastore.  This yang-data MUST be used if hints are to be
        provides back to the subscriber.";
      leaf reason {
        type identityref {
          base sn:modify-subscription-error;
        }
        description
          "Indicates the reason why the subscription has failed to
          be modified.";
      }
      uses hints;
    }
  }

  /*
   * NOTIFICATIONS
   */

  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 that would be subscribable as part of the NETCONF
```

```
       event stream by any receiver.";
    leaf subscription-id {
      type sn:subscription-id;
      description
        "This references the subscription which drove the notification
         to be sent.";
    }
    leaf incomplete-update {
      type empty;
      description
        "This is a flag which indicates that not all datastore nodes
         subscribed to are included with this update. In other words,
         the publisher has failed to fulfill its full subscription
         obligations, and despite its best efforts is providing an
         incomplete set of objects.";
    }
    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 snapshot corresponds to the same
         snapshot that would be returned in a corresponding get
         operation with the same selection 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;
      description
        "This references the subscription which drove the notification
         to be sent.";
    }
    leaf incomplete-update {
      type empty;
      description
        "The presence of this object indicates 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.";
```

```
    }
    anydata datastore-changes {
      description
        "This contains the set of datastore changes of the
         target datastore starting at the time of the
         previous update, per the terms of the subscription.
         The datastore changes are encoded per RFC 8027
         (YANG Patch).";
    }
  }

  augment "/sn:subscription-started" {
    description
      "This augmentation adds datastore-specific objects to
       the notification that a subscription has started.";
    uses update-policy;
  }

  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 {
          refine "selection-filter/within-subscription" {
          description
            "Specifies the selection filter and where it originated
            from. If the 'selection-filter-ref' is populated,
            the filter within the subscription came from the 'filters'
            container.  Otherwise it is populated in-line as part of the
            subscription itself.";
        }
      }
    }
  }

  augment "/sn:subscription-modified" {
    description
      "This augmentation adds datastore-specific objects to
       the notification that a subscription has been modified.";
    uses update-policy;
  }

  augment "/sn:subscription-modified/sn:target" {
    description
      "This augmentation allows the datastore to be included as part
      of the notification that a subscription has been modified.";
    case datastore {
```

```
      uses datastore-criteria {
         refine "selection-filter/within-subscription" {
         description
           "Specifies where the selection filter, and where it came
           from within the subscription and then populated within this
           notification. If the 'selection-filter-ref' is populated,
           the filter within the subscription came from the 'filters'
           container.  Otherwise it is populated in-line as part of the
           subscription itself.";
       }
     }
   }
 }

 /*
  * DATA NODES
  */

 augment "/sn:filters" {
   description
     "This augmentation allows the datastore to be included as part
      of the selection filtering criteria for a subscription.";
   list selection-filter {
     key "filter-id";
     description
       "A list of pre-configured filters that can be applied
        to datastore subscriptions.";
     leaf filter-id {
       type string;
       description
         "An identifier to differentiate between selection filters.";
     }
     uses selection-filter-types;
   }
 }

 augment "/sn:subscriptions/sn:subscription" {
   description
     "This augmentation adds many datastore specific objects to a
      subscription.";
   uses update-policy;
 }
 augment "/sn:subscriptions/sn:subscription/sn:target" {
   description
     "This augmentation allows the datastore to be included as part
     of the selection filtering criteria for a subscription.";
   case datastore {
      uses datastore-criteria;
```

```
    }
  }
}

<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-17.txt (RFC form)

7. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols 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 layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC5246].

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF 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:

First, it should be noted that the YANG module augments the YANG module from [I-D.draft-ietf-netconf-subscribed-notifications]. All security considerations that are listed there are relevant also for datastore subscriptions. In the following, we focus on the data nodes that are newly introduced here.

Subtree "selection-filter" under container "filters": This subtree allows to specify which objects or subtrees to include in a datastore subscription. An attacker could attempt to modify the filter. For example, the filter might be modified to result in very few objects being filtered in order to attempt to overwhelm the receiver. Alternatively, the filter might be modified to result in certain objects to be excluded from updates, in order to have certain changes go unnoticed.

Subtree "datastore" in choice "target" in list "subscription": Analogous to "selection filter", an attacker might attempt to modify the objects being filtered in order to overwhelm a receiver with a larger volume of object updates than expected, or to have certain changes go unnoticed.

Choice "update-trigger" in list "subscription": By modifying the update trigger, an attacker might alter the updates that are being sent in order to confuse a receiver, to withhold certain updates to be sent to the receiver, and/or to overwhelm a receiver. For example, an attacker might modify the period with which updates are reported for a periodic subscription, or it might modify the dampening period for an on-change subscription, resulting in greater delay of successive updates (potentially affecting responsiveness of applications that depend on the updates) or in a high volume of updates (to exhaust receiver resources).

RPC "resynch-subscription": This RPC allows a subscriber of an on-change subscription to request a full push of objects in the subscription’s scope. This can result in a large volume of data. An attacker could attempt to use this RPC to exhaust resources on the server to generate the data, and attempt to overwhelm a receiver with the resulting data volume.

8. Acknowledgments

For their valuable comments, discussions, and feedback, we wish to acknowledge Tim Jenkins, Martin Bjorklund, Kent Watsen, Susan Hares, Yang Geng, Peipei Guo, Michael Scharf, Guangying Zheng, and Tom Petch.

9. References

9.1. Normative References

[I-D.draft-ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Tripathy, A., and E. Nilsen-Nygaard, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-13 (work in progress), June 2018.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <https://www.rfc-editor.org/info/rfc6470>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://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, <https://www.rfc-editor.org/info/rfc7895>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8072] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Patch Media Type", RFC 8072, DOI 10.17487/RFC8072, February 2017, <https://www.rfc-editor.org/info/rfc8072>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

9.2. Informative References

[bergstra2014]
Bergstra, J. and M. Burgess, "Promise Theory, Createspace Independent Publishers, ISBN-13 9781495437779", 2014.

[I-D.draft-ietf-netconf-netconf-event-notifications]
Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "NETCONF Support for Event Notifications", May 2018.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7923] Voit, E., Clemm, A., and A. Gonzalez Prieto, "Requirements for Subscription to YANG Datastores", RFC 7923, DOI 10.17487/RFC7923, June 2016, <https://www.rfc-editor.org/info/rfc7923>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

Appendix A. Appendix A: Subscription Errors

A.1. RPC Failures

Rejection of an RPC for any reason is indicated by via RPC error response from the publisher. Valid RPC errors returned include both existing transport layer RPC error codes, such as those seen with NETCONF in [RFC6241], as well as subscription specific errors such as those defined within the YANG model. As a result, how subscription errors are encoded within an RPC error response is transport dependent.

References to specific identities within the either the subscribed-notifications YANG model or the yang-push YANG model may be returned as part of the error responses resulting from failed attempts at datastore subscription. Following are valid errors per RPC (note: throughout this section the prefix 'sn' indicates an item imported from the subscribed-notifications.yang model):

```
establish-subscription         modify-subscription
-----------------------        -------------------
 cant-exclude                   sn:filter-unsupported
 datastore-not-subscribable     sn:insufficient-resources
 sn:dscp-unavailable            sn:no-such-subscription
 sn:filter-unsupported          period-unsupported
 sn:insufficient-resources      result-too-big
 on-change-unsupported          synchronization-size
 on-change-synch-unsupported    unchanging-selection
 period-unsupported
 result-too-big                 resynch-subscription
 synchronization-size           --------------------
 unchanging-selection            no-such-subscription-resynch
                                 synchronization-size

delete-subscription            kill-subscription
-----------------------        -----------------
 sn:no-such-subscription        sn:no-such-subscription
```

There is one final set of transport independent RPC error elements included in the YANG model. These are the following four yang-data structures for failed datastore subscriptions:

1. yang-data establish-subscription-error-datastore
   This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "establish-subscription" RPC response. This MUST be sent if hints are included.

2. yang-data modify-subscription-error-datastore
   This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "modifiy-subscription" RPC response. This MUST be sent if hints are included.

3. yang-data sn:delete-subscription-error
   This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "delete-subscription" or "kill-subscription" RPC response.

4. yang-data resynch-subscription-error
   This MUST be returned if information identifying the reason for an RPC error has not been placed elsewhere within the transport portion of a failed "resynch-subscription" RPC response.

A.2. Notifications of Failure

A subscription may be unexpectedly terminated or suspended independent of any RPC or configuration operation. In such cases, indications of such a failure MUST be provided. To accomplish this, the following types of error identities may be returned within the corresponding subscription state change notification:

subscription-terminated	subscription-suspended
datastore-not-subscribable	sn:insufficient-resources
sn:filter-unavailable	period-unsupported
sn:no-such-subscription	result-too-big
sn:suspension-timeout	synchronization-size
unchanging-selection	

Appendix B. Changes between revisions

(To be removed by RFC editor prior to publication)

v16 - v17

- o Minor updates to YANG module, incorporating comments from Tom Petch.

- o Updated references.

v15 - v16

- o Updated security considerations.
- o Updated references.
- o Addressed comments from last call review, specifically comments received from Martin Bjorklund.

v14 - v15

- o Minor text fixes. Includes a fix to on-change update calculation to cover churn when an object changes to and from a value during a dampening period.

v13 - v14

- o Minor text fixes.

v12 - v13

- o Hint negotiation models now show error examples.
- o yang-data structures for rpc errors.

v11 - v12

- o Included Martin’s review clarifications.
- o QoS moved to subscribed-notifications
- o time-of-update removed as it is redundant with RFC5277’s eventTime, and other times from notification-messages.
- o Error model moved to match existing implementations
- o On-change notifiable removed, how to do this is implementation specific.
- o NMDA model supported. Non NMDA version at https://github.com/netconf-wg/yang-push/

v10 - v11

- o Promise model reference added.

- o Error added for no-such-datastore

- o Inherited changes from subscribed notifications (such as optional feature definitions).

- o scrubbed the examples for proper encodings

v09 - v10

- o Returned to the explicit filter subtyping of v00-v05

- o identityref to ds:datastore made explicit

- o Returned ability to modify a selection filter via RPC.

v08 - v09

- o Minor tweaks cleaning up text, removing appendicies, and making reference to revised-datastores.

- o Subscription-id optional in push updates, except when encoded in RFC5277, Section 4 one-way notification.

- o Finished adding the text descibing the resynch subscription RPC.

- o Removed relationships to other drafts and future technology appendicies as this work is being explored elsewhere.

- o Deferred the multi-line card issue to new drafts

- o Simplified the NACM interactions.

v07 - v08

- o Updated YANG models with minor tweaks to accommodate changes of ietf-subscribed-notifications.

v06 - v07

- o Clarifying text tweaks.

- o Clarification that filters act as selectors for subscribed datastore 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 [subscribe] 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
VMware

Email: agonzalezpri@vmware.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
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

B. Lengyel
Ericsson
A. Clemm
Huawei USA

July 2, 2018

YangPush Notification Capabilities
draft-lengyel-netconf-notification-capabilities-02

Abstract

This document proposes a YANG module that allows a YANG server to specify for which data nodes it will send "YANG Datastore Subscription" on-change notifications. It also proposes to use YANG Instance Data to document this information in implementation time.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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. Terminology 2
2. Introduction 2
3. On-change Notification Capability Model 4
3.1. Tree Diagram 5
3.2. YANG Module 5
4. Security Considerations 7
5. IANA Considerations 8
5.1. The IETF XML Registry 8
5.2. The YANG Module Names Registry 8
6. Open Issues 8
7. References 8
7.1. Normative References 8
7.2. Informative References 9
Appendix A. Changes between revisions 9
Authors' Addresses 9

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.

On-change Notification Capability: The capability of the YANG server to send on-change notifications on the change of the value for a specific data node.

Implementation-time information: Information about the YANG server's behavior that is made available during the implementation of the server, available from a source other then a running Yang server.

Rutime-information: Information about the YANG server's behavior that is available from the running YANG server via a protocol like NETCONF, RESTCONF or HTTPS.

2. Introduction

As defined in [I-D.ietf-netconf-yang-push] a YANG server may allow clients to subscribe to updates from a datastore and subsequently push such update notifications to the client. Notifications may be sent periodically or on-change (more or less immendiately after each change).

In some cases, a publisher supporting on-change notifications will not be able to push updates for some object types on-change. Reasons

for this might be that the value of the datastore node changes frequently (e.g. 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. In those cases, it will be important for client applications to have a way to identify for which objects on-change notifications are supported and for which ones are not supported.

Faced with the reality that support for on-change notification does not mean that such notifications will be sent for any specific data node, client/management applications can not rely on the on-change functionality unless the client has some means to identify for which objects on-change notifications are supported and for which ones are not supported. YANG models are meant to be used as an interface contract. Without identification of data nodes supporting on-change, this contract would only state the YANG server may (or may not) send on-change notifications for a data node specified in a YANG module.

This document proposes a YANG module that allows a client to identify which data nodes support on-change notification, removing the uncertainty for on-change notifications.

On-change Notification Capability information will be needed both in implementation-time and run-time.

Run-time information is needed

- o for any "purely model driven" client, e.g. a Netconf-browser. As long as it has a valid model, it does not care which data nodes send notification, it will just handle whats available.

- o to check that early implementation time information about the capability is indeed what the server supports

- o in case the capability might change during run-time e.g. due to licensing, HW constraints etc.

Implementation time information is needed by Network Management System (NMS) implementers. During NMS implementation for any functionality that depends on the notifications the information about on change notification capability is needed. If the information is not available early in some document, but only as instance data from the network node, the NMS implementation will be delayed, because it has to wait for the network node to be ready. Also assuming that all NMS implementers will have a correctly configured network node available to retrieve data from, is an expensive proposition. (An NMS may handle dozens of network node types.) Often a fully

functional NMS is a requirement for introducing a new network node type into a network, so delaying the NMS effectively delays the availability of the network node as well.

Implementation time information is needed by system integrators. System integrators will need information about on change notification capability early. When introducing a network node type into their network operators often need to integrate the node type into their own management system. The NMS may have management functions that depend on on-change notifications. The network operator needs to plan his management practices and NMS implementation before he even decides to buy the specific network node type. Moreover the decision to buy the node type sometimes depends on these management possibilities.

3. On-change Notification Capability Model

As described above a number of stakeholders need information about the on change notification capability both in implementation and run-time. It is a goal to provide this information in a format that is

- o vendor independent (standard)
- o formal (no freeform English text please)
- o the same both in implementation-time and run-time

The YANG module ietf-notification-capabilities is defined to provide information about the on-change notification capabilities. There is a default notification capability separately for config false and config true data nodes. There is also an on-change-notification-capability list containing a potentially different true/false notification capability for any data node in the schema tree. Unless a node is in the list with a specific capability value, it inherits its on-change-notification-capability from its parent in the data tree, or from the relevant default values.

The instance information SHALL be provided in two ways both following the ietf-notification-capabilities module:

- o It SHALL be provided by the implementer as YANG instance data file complying to the [I-D.lengyel-netmod-yang-instance-data]. The file SHALL be available already in implementation time retrievable in a way that does not depend on a live network node. E.g. download from product Website.
- o It SHALL be available via Netconf or Restconf from the live YANG server during runtime.

3.1. Tree Diagram

The following tree diagram [RFC8340] provides an overview of the data model.

```
module: ietf-notification-capabilities
    +--ro on-change-notification-capability
       +--ro notification-sent-for-config-default?   boolean
       +--ro notification-sent-for-state-default?    boolean
       +--ro on-change-notification-capability* [node-selector]
          +--ro node-selector             nacm:node-instance-identifier
          +--ro on-change-notification-sent    boolean
```

3.2. YANG Module

<CODE BEGINS> file "ietf-notification-capabilities.yang"

```
module ietf-notification-capabilities {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-notification-capabilities";
  prefix inc;

  import ietf-netconf-acm { prefix nacm; }

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

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

     WG Chair: Kent Watsen
               <mailto:kwatsen@juniper.net>

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

     Editor:   Balazs Lengyel
               <mailto:balazs.lengyel@ericsson.com>";


  description "This module specifies for which data nodes will the
    YANG server send on-change notifications.

    On-change notification capability is marked as true or false.
    This marking is inherited from the parent down the data tree
    unless explicitly marked otherwise.
```

```
    On-change notifications SHALL be sent for a config=true
    data node if one of the following is true:
    - it is specified in the on-change-notification-capability
    list and has a on-change-notification-sent value true or
    - notifications are sent for its parent data node and it is
    not specified in the on-change-notification-capability list or
    - if it is a top level data-node and is not specified in the
    on-change-notification-capability list and the
    notification-sent-for-config-default is true.

    On-change notifications SHALL be sent for a config=false
    data node if one of the following is true:
    - it is specified in the on-change-notification-capability
    list and has an on-change-notification-sent value true or
    - notifications are sent for its parent data node
    which is also config=false and it is
    not specified in the on-change-notification-capability list or
    - if it is a top level data-node or has a config=true parent
    data node and is not specified in the
    on-change-notification-capability list and the
    notification-sent-for-state-default is true.
    ";

    reference "RFC XXXX Yang-Push";

  revision 2018-07-02 {
    description "Initial version";
    reference
      "RFC XXX: YangPush Notification Capabilities";
  }

  container on-change-notification-capability {
    config false;
    description "Contains default values for the
      on-change-notifiction-capability and a list of data nodes that
      have the on-change-notification-capability specifically defined.";

    leaf notification-sent-for-config-default {
      type boolean;
      default true;
      description "Specifies the default value for
        top level configuration data nodes for the
        on-change-notification-sent capability.";
    }

    leaf notification-sent-for-state-default {
      type boolean;
      default false;
```

```
      description "Specifies the default value
        top level state data nodes for the
        on-change-notification-sent capability.";
    }

    list on-change-notification-capability {
      key node-selector ;
      description "A list of data nodes that have the
         on-change-notification-capability specifically defined.

         Should be used when specific data nodes support
         on-change notification in a module/subtree that
         generally does not support it or when some data nodes
         do not support the notification in a module/subtree
         that generally supports on-change notifications.";

      leaf node-selector {
        type nacm:node-instance-identifier;
      }

      leaf on-change-notification-sent {
        type boolean;
        mandatory true;
        description "Specifies whether the YANG server will
          send on-change notifications for the selected
          data nodes.";
      }
    }
  }
}
```

<CODE ENDS>

4. Security Considerations

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

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

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 registrations are requested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-notification-capabilities
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 [RFC7950]. Following the format in [RFC7950], the the following registrations are requested:

```
name:       ietf-notification-capabilities
namespace:  urn:ietf:params:xml:ns:yang:ietf-notification-capabilities
prefix:     inc
reference:  RFC XXXX
```

6. Open Issues

```
Do we need separate defaults/individual lists for every datastore?
Proposal: no, it would be an overkill.

Should type nacm:node-instance-identifier be moved to yang-types?
It is useful for more then just nacm.
```

7. References

7.1. Normative References

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", draft-ietf-netconf-yang-push-17 (work in progress), July 2018.

[I-D.lengyel-netmod-yang-instance-data]
Lengyel, B. and B. Claise, "YANG Instance Data Files and their use for Documenting Server Capabilities", draft-lengyel-netmod-yang-instance-data-02 (work in progress), July 2018.

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

7.2. Informative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Changes between revisions

v01 - v02

- o Instead of augmenting ietf-yang-library a more simple standalone model is proposed.

v00 - v01

- o Corrections
- o Augment only the new yanglib
- o Correct the condtions for notifying state data

Authors' Addresses

Balazs Lengyel
Ericsson
Magyar Tudosok korutja 11
1117 Budapest
Hungary

Email: balazs.lengyel@ericsson.com

Alexander Clemm
Huawei USA
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: ludwig@clemm.org

NETMOD Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 15, 2018

L. Lhotka
CZ.NIC

June 13, 2018

RESTCONF with Transactions
draft-lhotka-netconf-restconf-transactions-00

Abstract

This document extends the RESTCONF protocols with transaction capabilities that allow for safe concurrent access of multiple clients.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 15, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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
2.1. YANG 3
2.2. HTTP 3
2.3. RESTCONF 3
2.4. NMDA 3
2.5. New Terms 4
3. Datastores 4
3.1. The Staging Configuration Datastore 4
4. New Operations 5
4.1. Commit 5
4.2. Reset 5
5. Access Control 5
6. Compatibility 6
7. YANG Module 6
8. IANA Considerations 8
9. Security Considerations 8
10. References 8
10.1. Normative References 8
10.2. URIs 10
Author’s Address 10

1. Introduction

The RESTCONF protocol [RFC8040] was introduced as a simpler alternative to the original NETCONF protocol [RFC6241]. Due to the simplicity requirement, some more complex features and functions of NETCONF, such as locks, subtree filtering or candidate configuration datastore, are not available in RESTCONF.

On the other hand, RESTCONF offers several advantages over NETCONF, including:

- o the use of HTTP methods and well-known Representational State Transfer (REST) approaches make it more accessible to developers and increases the choice of software libraries and tools

- o cleaner semantics of edit operations,

- o alternative encodings in which resources can be represented, currently JSON and XML; NETCONF supports only XML

- o certain HTTP mechanisms, such as "Last-Modified" and "ETag" headers.

This document extends the RESTCONF protocol with transaction capabilities, at the cost of adding two RPC operations and some complexity in the server implementation. This makes RESTCONF suitable for network management environments where concurrent access of multiple client is needed.

A RESTCONF server indicates support for transactions as defined in this document by including the YANG module "ietf-restconf-transactions" (Section 7) among implemented modules in its YANG library data [I-D.ietf-netconf-rfc7895bis].

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 [RFC2119].

2.1. YANG

The following terms are defined in [RFC7950]:

- o RPC operation

2.2. HTTP

The following terms are defined in [RFC7231]:

- o resource

The following terms are defined in [RFC7232]:

- o entity-tag

2.3. RESTCONF

The following terms are defined in [RFC8040]:

- o client
- o RESTCONF root resource, {+restconf}

2.4. NMDA

The following terms are defined in [RFC8342]:

- o candidate configuration datastore, <candidate>
- o intended configuration datastore, <intended>

- o operational state datastore, <operational>

- o running configuration datastore, <running>

2.5. New Terms

The following new term is used in this document:

- o staging configuration datastore: a configuration datastore that represents a staging area private to each RESTCONF user, and that is eventually commited into <intended>.

3. Datastores

A RESTCONF server implementing this document MUST be NMDA-compliant [I-D.ietf-netconf-nmda-restconf]. Apart from the operational state datastore, it MUST also support the intended configuration datastore.

The intended configuration datastore SHOULD persist across server reboots. In terms of the NMDA architecture [RFC8342], <intended> can be considered identical to <running>, although this document does not explicitly use the latter datastore.

3.1. The Staging Configuration Datastore

This document introduces a new configuration datastore named ’staging’ that represents a staging area private to each user (as identified by RESTCONF username, see [RFC8040] Section 2.5).

In NETCONF terms ([RFC8040] Section 8.3), the staging datastore is essentially a non-shared candidate configuration datastore. The new name is used in order to emphasize the narrower semantics: the staging datastore MUST be private to each user.

Note that the above requirement does not necessarily mean that each user is provided with a separate copy of configuration data. For instance, several efficient methods utilizing persistent data structures and copy-on-write are available. However, the particular implementation approach is outside the scope of this document.

The staging datastore assumes the place of the datastore resource as defined in [RFC8040] Section 3.4. This means that all resources inside the "{+restconf}/data" subtree correspond to data instances in the staging datastore. Therefore, the contents of the staging datastore can be retrieved by means of the GET method and modified by means of PUT, POST and PATCH methods exactly as specified in [RFC8040]. However, the changes to the staging datastore MUST NOT

impact operational state of the device until they are merged into <intended> via the "commit" operation (Section 4.1).

4. New Operations

In order to support transactions in RESTCONF, the YANG module "ietf-restconf-transactions" defines two RPC operations described below.

4.1. Commit

The "commit" operation atomically merges the contents of the client's staging datastore into <intended>.

The concrete strategy and implementation of the merge procedure is outside the scope of this document. The resolution of merge conflicts can be fully automatic, which is preferable, or may require client's intervention. In the latter case, the server SHALL send an error response with the following properties:

- o HTTP status code 412
- o error-tag of "operation-failed"
- o error-app-tag of "merge-conflict"
- o error-info containing additional information to aid the user in resolving the conflict.

4.2. Reset

The "reset" operation resets the user's staging datastore to the current contents of <intended>.

If the server supports entity-tags (see [RFC8040] Section 3.5.2), then after completing the "reset" operation the entity-tags for the staging and intended datastore resources MUST be identical.

5. Access Control

A server that implements this document along with NETCONF Access Control Model [RFC8341] MUST guarantee that all NACM rules are observed. This means in particular:

- o Configuration data that is not readable for a given user MUST NOT be exposed in the user's staging datastore.

- o A commit operation executed by a given user MUST NOT modify configuration data in <intended> in a way that is not compliant with NACM rules that are in effect for that user.

6. Compatibility

RESTCONF with transactions, as defined in this document, is compatible with the original RESTCONF specification [RFC8040] and RESTCONF NMDA extensions [I-D.ietf-netconf-nmda-restconf].

In practical terms, standard RESTCONF clients are able to retrieve and edit data in the staging configuration datastore. If they support NMDA, they can also retrieve data from <intended> and <operational>. In order to make them useful for network management, it is only necessary to allow for executing the "commit" and "reset" operations. This can be accomplished through simple scripts utilizing curl [1] or similar tools.

7. YANG Module

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

```
<CODE BEGINS> file "ietf-restconf-transactions@2018-06-11.yang"

module ietf-restconf-transactions {

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-restconf-transactions";

  prefix rct;

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

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

     WG Chair: Kent Watsen
               <mailto:kwatsen@juniper.net>

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

     Editor:   Ladislav Lhotka
               <mailto:lhotka@nic.cz>";
```

```
     description
       "This module defines operations that implement transactions in
        the RESTCONF protocol.

        Copyright (c) 2018 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
        (https://trustee.ietf.org/license-info).

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and
        'OPTIONAL' in the module text are to be interpreted as described
        in RFC 2119 (https://tools.ietf.org/html/rfc2119).

        This version of this YANG module is part of RFC XXXX
        (https://tools.ietf.org/html/rfcXXXX); see the RFC itself for
        full legal notices.";

     revision 2018-06-11 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: RESTCONF with Transactions";
     }

     /* Operations */

     rpc commit {
       description
         "Atomically merge the contents of client's staging datastore
          into the intended datastore.

          If a merge conflict occurs that cannot be automatically
          resolved, the server SHALL send an error report with
          error-app-tag set to 'merge-conflict' and error-info
          indicating the reason for the conflict.";
     }

     rpc reset {
       description
         "Reset the client's staging datastore so that its contents is
          identical to the contents of the intended repository.";
     }
```

```
}

<CODE ENDS>
```

8. IANA Considerations

RFC Editor: In this section, replace all occurrences of 'XXXX' with the actual RFC number and all occurrences of the revision date below with the date of RFC publication (and remove this note).

This document registers one URI in the IETF XML Registry [RFC3688]. The following registration has been made:

URI: urn:ietf:params:xml:ns:yang:ietf-restconf-transactions

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers one YANG module in the YANG Module Names Registry [RFC6020]. The following registration has been made:

```
name:       ietf-restconf-transactions
namespace:  urn:ietf:params:xml:ns:yang:ietf-restconf-transactions
prefix:     rct
reference:  RFC XXXX
```

9. Security Considerations

TBD

10. References

10.1. Normative References

[I-D.ietf-netconf-nmda-restconf]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "RESTCONF Extensions to Support the Network Management Datastore Architecture", draft-ietf-netconf-nmda-restconf-04 (work in progress), April 2018.

[I-D.ietf-netconf-rfc7895bis]
Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", draft-ietf-netconf-rfc7895bis-06 (work in progress), April 2018.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <https://www.rfc-editor.org/info/rfc7232>.

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

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

10.2. URIs

[1] https://curl.haxx.se

Author's Address

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

NETCONF WG
Internet-Draft
Updates: 6241 (if approved)
Intended status: Standards Track
Expires: January 1, 2019

M. Jethanandani
J. Lam
A. Leung
Cisco Systems, Inc.
A. Bierman
YumaWorks, Inc.
June 30, 2018

Binary Encoding for NETCONF
draft-mahesh-netconf-binary-encoding-01

Abstract

This document describes a method by which a NETCONF [RFC6241] client and server can negotiate an alternate form of encoding.

This document updates RFC 6241.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 1, 2019.

Copyright Notice

Copyright (c) 2018 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 (https://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
2. Protocol Negotiation 3
2.1. Encoding 3
2.1.1. Overview 3
2.1.2. Example 4
2.1.3. Dependencies 5
2.1.4. Capability Identifier 5
2.1.5. New Operations 6
2.1.6. Modification to Existing Operations 6
2.1.7. Interactions with Other Capabilities 6
2.2. CBOR and SID 6
3. Security Considerations 6
4. IANA Considerations 6
4.1. NETCONF Capability URNs 6
4.2. New Registry 7
5. Acknowledgements 7
6. References 7
6.1. Normative References 7
6.2. Informative References 8
Authors' Addresses 8

1. Introduction

NETCONF [RFC6241], by default, supports XML encoding for its payload. However, XML can be very verbose, specially for operational data. This document proposes a mechanism by which clients and servers can negotiate an alternate form of encoding, e.g. binary encoding, and use that to exchange data. Binary encoding can reduce the physical size of the data exchanged, in some cases by as much as 66%, while preserving the original data.

Several encoding mechanisms have been proposed, including CBOR [RFC7049] and JSON [RFC8259]. This document does not advocate any particular encoding format. Instead, it leaves it up to the negotiation between client and server to decide the form of encoding. For an example of how to encode YANG in CBOR format, see CBOR Encoding of Data Modeled with YANG [I-D.ietf-core-yang-cbor] and JSON Encoding of Data Modeled with YANG [RFC7951].

This document updates NETCONF [RFC6241].

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.

2. Protocol Negotiation

NETCONF clients and servers exchange a hello as part of establishing a connection. As part of the hello exchange, the client advertises an ordered list of encoding it would like to use, while the server advertises an unordered list of encoding that it is capable of supporting. If no match of encoding is found, the session is dropped. If a match is found the client issues a request that is encoded with first match found. Thereafter, both the Message layer in Figure 1 of NETCONF [RFC6241] and the YANG data within the Message Layer, are in the form of new encoding. This includes RPC, Actions and Notifications.

This draft suggests advertisement of the following additional capability.

2.1. Encoding

2.1.1. Overview

The :encoding capability indicates what alternate encoding format each side is willing to support. The client and server send a comma separated list (with no white spaces) of encoding formats they are willing to support. The client sends a list of encoding ordered by preference, while the server includes an unordered list of encoding.

Both the client and server examine each others <hello> message for this capability. If not present, the default encoding is used, which is XML. The client examines its list against the server list, checked in the order of preference it sent do the server. If a matching encoding format is found, the client picks that encoding for the remainder of the session, starting with the first <rpc> request. All <rpc>, <rpc-reply>, <action> and <notification> messages MUST be encoded in this negotiated encoding.

Both the client and the server MUST support the "application/xml" media type to be backward compatible with NETCONF [RFC6241].

The base:1.1 negotiation defined in NETCONF [RFC6241] determines the message framing that is used for the entire session.

2.1.2. Example

In this example, the client supports the following encoding formats shown in a preferred order.

- o Concise Binary Object Representation (CBOR) with YANG Schema Item iDentifier (SID) - cbor+sid

- o Google Protocol Buffers (GPB) - gpb

- o Thrift - thrift

In this example, the client advertises its (abbreviated) <hello> as follows. Some extra white spaces have been added for display purposes only.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>
      urn:ietf:params:netconf:capability:encoding:1.0?format=
      application/cbor+sid,application/gpb,application/thrift
    </capability>
  </capabilities>
</hello>
```

The server supports the following encoding formats shown in no particular order of preference.

- o cbor+sid

- o gpb

In this example, the server advertises its (abbreviated) <hello> as follows. Some white spaces have been added for display purposes only.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>
      urn:ietf:params:netconf:capability:encoding:1.0?
      format=application/cbor+sid,application/gpb
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:config-id?id=2130
    </capability>
    <!--  rest of URIs ... -->
  </capabilities>
  <session-id>4</session-id>
</hello>
```

The common encoding formats in both the list are "application/cbor+sid", and "application/gpb", but since cbor+sid appear first on the client list, "application/cbor+sid" is selected as the form of encoding for the remainder of the session.

2.1.3. Dependencies

None.

2.1.4. Capability Identifier

The :encoding capability is identified by the following capability string:

urn:ietf:params:netconf:capability:encoding:1.0?format={name, ...}

The :encoding capability MUST be advertised in every server <hello> message and the URI MUST contain a "format" argument assigned a comma-separated list (with no white spaces) of formats supported by the device. For the list of supported formats, this document requests the creation of a new registry. See IANA Considerations for details.

The client on the other hand SHOULD advertise this capability in its <hello> message, but it MAY omit it if XML encoding is desired.

For example (line wrapped for display purposes only)

urn:ietf:params:netconf:capability:encoding:1.0?format=
application/cbor+sid,application/gpb,application/thrift

2.1.5. New Operations

The :encoding capability does not introduce any new protocol operation.

2.1.6. Modification to Existing Operations

The :encoding capability does not modify any existing protocol operations.

2.1.7. Interactions with Other Capabilities

The :encoding capability does not interact with any other capabilities.

2.2. CBOR and SID

One of the encoding formats that can be advertised by the client or the server is CBOR [RFC7049]. The payload consists of YANG [RFC7950] data, and YANG requires the use of unique identifiers, implemented in NETCONF [RFC6241] using names. To allow for encoding of YANG data models, a more compact method has been identified, called YANG Schema Item iDentifier (SID) [I-D.ietf-core-yang-cbor]. Clients and servers can advertise their capability for this form of encoding using "application/cbor+sid".

SID does not define encoding for NETCONF operations today. It is expected that a new SID range would have to be identified for NETCONF protocol operations.

3. Security Considerations

4. IANA Considerations

This document registers a URI and requests the creation of a new registry.

4.1. NETCONF Capability URNs

This document requests the registry of an URI in the IETF XML registry [RFC3688]. The IANA registry "Network Configuration Protocol (NETCONF) Capability URNs" needs to be updated to include the following capability.

```
Index
    Capability Identifier
-------------------------
:encoding
    urn:ietf:params:netconf:capability:encoding:1.0
```

4.2. New Registry

The document also requests the creation of a new registry, called "Network Configuration Protocol (NETCONF) Encoding formats", that should be populated with the following entries.

```
Encoding Formats
----------------
cbor+sid
gpb
thrift
```

5. Acknowledgements

The authors would like to thank Juergen Schoenwaeider for his comments on the draft.

6. References

6.1. Normative References

[I-D.ietf-core-yang-cbor] Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-06 (work in progress), February 2018.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://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, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

6.2. Informative References

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

Authors' Addresses

Mahesh Jethanandani

Email: mjethanandani@gmail.com

Jason Lam
Cisco Systems, Inc.

Email: lamj@cisco.com

Alfred Leung
Cisco Systems, Inc.

Email: alfleung@cisco.com

Andy Bierman
YumaWorks, Inc.

Email: andy@yumaworks.com

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 21, 2018

Q. Wu
R. Ranade
Huawei

June 19, 2018

Base Notifications for NMDA
draft-wu-netconf-base-notification-nmda-01

Abstract

The Network Configuration Protocol (NETCONF) provides mechanisms to manipulate configuration datastores. NMDA introduces additional datastores for systems that support more advanced processing chains converting configuration to operational state. However, client applications are not able to be aware of common events pertaining to additional datstores, such as a data validation state change in NETCONF server, that may impact management applications. This document updates [RFC6470] to allow a NETCONF client to receive additional notifications for some common system events pertaining to the Network Management Datastore Architecture (NMDA) defined in [RFC8342].

Status of This Memo

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

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

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

This Internet-Draft will expire on December 21, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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
2. Summary of Updates to RFC 6470 3
3. NETCONF Base Notifications YANG Model extension for NMDA . . 3
3.1. Overview 3
3.2. Definitions 5
4. Security Considerations 8
5. IANA Considerations 8
6. Acknowledgements 9
7. Contributors 9
8. Normative References 9
Authors' Addresses 10

1. Introduction

The Network Configuration Protocol (NETCONF) [RFC6470] provides mechanisms to manipulate configuration datastores. NMDA introduces additional datastores (e.g., <intended>, <operational>) for systems that support more advanced processing chains converting configuration to operational state. However, client applications are not able to be aware of common events pertaining to additional datastores, e.g., there are many background activities that happen during the time that configuration is committed to <running> to the time that the configuration is actually applied to <operational>. It is possible that some configuration could not be applied to <operational> due to either validation issues, or missing resource etc. There is a need for user to know the validation result of <intended> data-store and the reason why the configuration were not applied.

This document updates [RFC6470] to allows a NETCONF client to receive additional notifications for some common system events pertaining to the Network Management Datastore Architecture (NMDA) defined in [RFC8342]. These notification are not specific to any network management protocols such as NETCONF and RESTCONF.

The solution presented in this document is backwards compatible with [RFC6470].

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.

The following terms are defined in [RFC8342] and are not redefined here:

- o operational state datastore
- o running configuration datastore
- o intended configuration datastore

2. Summary of Updates to RFC 6470

This document is intended to provide an extension of notifications initially defined within [RFC6470], with the development of NMDA architcture and data model. Key relationships between these two documents include:

- o The existing notifications defined in [RFC6470] are remain unchanged, no additional information is added.
- o A new event notification is defined in this document to overcome the shortcoming of [RFC6470] for supporting NMDA.

3. NETCONF Base Notifications YANG Model extension for NMDA

3.1. Overview

The YANG module in NETCONF Base Notifications [RFC6470] specifies the following 5 event notifications for the ’NETCONF’ stream to notify a client application that the NETCONF server state has changed:

- o netconf-config-change
- o netconf-capability-change
- o netconf-session-start
- o netconf-session-end
- o netconf-confirmed-commit

These event notifications used within the 'NETCONF' stream are accessible to clients via the subscription mechanism described in [RFC5277].

This document extends the YANG module defined in [RFC6470] to include NMDA specific extension which allows a NETCONF client to receive notifications for additional common system event as follows:

nmda-data-validate: Generated when a server with network management protocol support detects that a data validation event has occurred from the time that configuration is committed to <running> to the time that the configuration is actually applied to <operational> during management session. Indicates the event and the current state of the data validation. A server MAY report events for non-NETCONF management sessions (such as RESTCONF,gPRC), using the 'session-id' value of zero.

These notification messages are accessible to clients via either the subscription mechanism described in [RFC5277] or dynamic subscription mechanism and configured subscription mechanism described in [I-D.ietf-netconf-netconf-event-notifications].

The following are examples of a nmda-data-validation notification message:

```
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
 <eventTime>2017-06-16T16:30:59.137045+09:00</eventTime>
 <nmda-data-validate xmlns="urn:ietf:params:xml:ns:yang:ietf-nmda-notifications"
>
     <username>admin</username>
     <session-id>0</session-id>
     <source-host>10.251.93.83</source-host>
     <validate-event>start<validate-event>
     <validate-result>partial-fail</validate-result>
     <validate-fail-taget>
      <datastore>intended</datastore>
      <target> /ietf-interfaces:interfaces-state </target>
     </validate-fail-target>
     <validate-fail-taget>
      <datastore>intended</datastore>
      <target> /ietf-system:system </target>
     </validate-fail-target>
 </nmda-data-validate>
</notification>
```

3.2. Definitions

This section presents the YANG module defined in this document. This module imports data types from the 'ietf-netconf' module defined in [RFC6241] and 'ietf-inet-types' module defined in [RFC6021].

```
<CODE BEGINS> file "ietf-nmda-notifications@2018-04-01.yang"
module ietf-nmda-notifications {
  namespace "urn:ietf:params:xml:ns:yang:ietf-nmda-notifications";
  prefix nmdan;

  import ietf-datastores {
    prefix ds;
  }
  import ietf-inet-types { prefix inet; }
  organization
    "IETF NETCONF (Network Configuration Protocol) Working Group";
  contact
    "WG Web:   <http://tools.ietf.org/wg/netconf/>
     WG List:  <mailto:netconf@ietf.org>
     WG Chair: Kent Watsen
               <mailto:kwatsen@juniper.net>
     WG Chair: Mahesh Jethanandani
               <mailto:mjethanandani@gmail.com>
     Editor:   Qin Wu
               <mailto:bill.wu@huawei.com>
     Editor:   Rohit R Ranade
               <mailto:rohitrranade@huawei.com>";
  description
    "This module defines a YANG data model for use with the
     NETCONF protocol that allows the NETCONF client to
     receive additional common NETCONF base event notifications
      related to NMDA.
     Copyright (c) 2012 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 version of this YANG module is part of RFC xxxx; see
     the RFC itself for full legal notices.";

  revision 2018-04-01 {
    description
      "Initial version.";
    reference "RFC xxx: NETCONF Base Notifications for NMDA";
```

```
    }
    typedef session-id-or-zero-type {
       type uint32;
       description
         "NETCONF Session Id or Zero to indicate none";
     }

    grouping common-session-parms {
      description
        "Common session parameters to identify a
         management session.";

      leaf username {
        type string;
        mandatory true;
        description
          "Name of the user for the session.";
      }

      leaf session-id {
        type session-id-or-zero-type;
        mandatory true;
        description
          "Identifier of the session.
           A NETCONF session MUST be identified by a non-zero value.
           A non-NETCONF session MAY be identified by the value zero.";
      }

      leaf source-host {
        type inet:ip-address;
        description
          "Address of the remote host for the session.";
      }
    }
    notification nmda-data-validate {
      description
        "Generated when a NETCONF server detects that a
         Data validation event has occurred.  Indicates the event
         and the current state of the data validation procedure
         in progress.";
      reference "RFC 8342, Section 5";
      uses common-session-parms;
      leaf validate-event {
        type enumeration {
          enum "start" {
            description
              "The data validate procedure has started.";
          }
```

```
          enum "complete" {
            description
              "The data validation procedure has been completed.";
          }
        }
        mandatory true;
        description
          "Indicates the event that caused the notification.";
      }
      leaf validate-result {
        when "../validate-event = 'complete'";
        type enumeration {
          enum "fail" {
            description
              "The <intended> configuration fails to be validated
               before being applied to <operational>, e.g., resources
               are not available or otherwise not physically present
               leads to the whole set of <intended>are not applied.";
          }
          enum "partial-fail" {
            description
              "The <intended> configuration partially fails to be
               validated before being applied to <operational>,e.g.,
               resources are not available or otherwise not physically
               present leads to parts of <intended>are not applied";
          }
          enum "success" {
            description
              "The <intended> configuration is successfully validated
               before being applied to <operational>.";
          }
        }
        description
          "Result of validate";
      }
      list fail-validate-target {
        leaf datastore {
          type identityref {
            base ds:datastore;
          }
          default "ds:operational";
          description
            "Indicates which datastore has changed or which datastore is
             target of edit-data operation.";
        }
        leaf target {
          type instance-identifier;
          description
```

```
            "Topmost node associated with the configuration change.
             A server SHOULD set this object to the node within
             the datastore that is being altered.  A server MAY
             set this object to one of the ancestors of the actual
             node that was changed, or omit this object, if the
             exact node is not known.";
        }
        description
          "List for fail validate targets";
      }
    }
  }
  <CODE ENDS>
```

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, defined in [RFC6242].

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:

/nmda-data-validate/validate-event:

Indicates the specific validate-event state change that occurred. A value of ’complete’ probably indicates that data validation procedure has completed.

5. IANA Considerations

This document registers one XML namespace URN in the ’IETF XML registry’, following the format defined in [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-nmda-notifications

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers one module name in the ’YANG Module Names’ registry, defined in [RFC7950]:

name: ietf-nmda-notifications

prefix: ncdn

namespace: urn:ietf:params:xml:ns:yang:ietf-nmda-notifications

RFC: xxxx

6. Acknowledgements

Thanks to Juergen Schoenwaelder, Alex Clemm,Carey Timothy and Andy Berman to review this draft and provide important input to this document.

7. Contributors

Xiaojian Ding
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: dingxiaojian1@huawei.com

8. Normative References

[I-D.ietf-netconf-netconf-event-notifications]
Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "NETCONF Support for Event Notifications", draft-ietf-netconf-netconf-event-notifications-09 (work in progress), May 2018.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", draft-ietf-netconf-yang-push-16 (work in progress), May 2018.

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <https://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, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6021, DOI 10.17487/RFC6021, October 2010, <https://www.rfc-editor.org/info/rfc6021>.

[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, <https://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, <https://www.rfc-editor.org/info/rfc6242>.

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, DOI 10.17487/RFC6470, February 2012, <https://www.rfc-editor.org/info/rfc6470>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

Authors' Addresses

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Rohit R Ranade
Huawei

Email: rohitrranade@huawei.com

NETCONF
Internet-Draft
Intended status: Standards Track
Expires: November 17, 2018

T. Zhou
G. Zheng
Huawei
E. Voit
Cisco Systems
A. Clemm
Huawei
A. Bierman
YumaWorks
May 16, 2018

Subscription to Multiple Stream Originators
draft-zhou-netconf-multi-stream-originators-02

Abstract

This document describes the distributed data collection mechanism that allows multiple data streams to be managed using a single subscription. Specifically, multiple data streams are pushed directly to the collector without passing through a broker for internal consolidation.

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 https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on November 17, 2018.

Copyright Notice

Copyright (c) 2018 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 (https://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. Use Cases 3
  2.1. Use Case 1: Data Collection from Devices with Main-board and Line-cards 3
  2.2. Use Case 2: IoT Data Collection 4
3. Solution Overview 5
4. Subscription Decomposition 7
5. Publication Composition 9
6. IANA Considerations 9
7. Security Considerations 9
8. Acknowledgements 9
9. References 9
  9.1. Normative References 9
  9.2. Informative References 10
Appendix A. Change Log 10
Appendix B. Subscription Management 11
Appendix C. Notifications on Subscription State Changes 11
Appendix D. Configured Subscription and Call Home 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.

YANG-Push [I-D.ietf-netconf-yang-push] defines a transport-independent subscription mechanism for datastore updates, in which a subscriber can subscribe to a stream of datastore updates from a server, or update provider. The current design involves subscription to a single push server. This conceptually centralized model encounters efficiency limitations in cases where the data sources are themselves distributed, such as line cards in a piece of network equipment. In such cases, it will be a lot more efficient to have each data source (e.g., each line card) originate its own stream of updates, rather than requiring updates to be tunneled through a central server where they are combined. What is needed is a distributed mechanism that allows to directly push multiple individual data substreams, 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 will describe such distributed data collection mechanism and how it can work by extending existing YANG-Push mechanism. The proposal is general enough to fit many scenarios.

2. Use Cases

2.1. Use Case 1: Data Collection from Devices with Main-board and Line-cards

For data collection from devices with main-board and line-cards, existing YANG-Push solutions consider only one push server typically reside in the main board. As shown in the following figure, data are collected from line cards and aggregate to the main board as one consolidated stream. So the main board can easily become the performance bottle-neck. The optimization is to apply the distributed data collection mechanism which can directly push data from line cards to a collector. On one hand, this will reduce the cost of scarce compute and memory resources on the main board for data processing and assembling. On the other hand, distributed data push can off-load the streaming traffic to multiple interface

```
                  +-------------------------------------+
                  |              collector              |
                  +------^-----------^-----------^------+
                         |           |           |
                  +-------------------------------------+
                  |      |           |           |      |
                  |      |     +-----+------+    |      |
                  |      |     | main board |    |      |
                  |      |     +--^-----^---+    |      |
                  |      |        |     |        |      |
                  |      |    +---+     +---+    |      |
                  |      |    |             |    |      |
                  | +----+----+---+     +---+----+----+ |
                  | | line card 1 |     | line card 2 | |
                  | +-------------+     +-------------+ |
                  |               device                |
                  +-------------------------------------+
```

Fig. 1 Data Collection from Devices with Main-board and Line-cards

2.2. Use Case 2: IoT Data Collection

In the IoT data collection scenario, as shown in the following figure, collector usually cannot access to IoT nodes directly, but is isolated by the border router. So the collector subscribes data from the border router, and let the border router to disassemble the subscription to corresponding IoT nodes. The border router is typically the traffic convergence point. It’s intuitive to treat the border router as a broker assembling the data collected from the IoT nodes and forwarding to the collector[I-D.ietf-core-coap-pubsub]. However, the border router is not so powerful on data assembling as a network device. It’s more efficient for the collector, which may be a server or even a cluster, to assemble the subscribed data if possible. In this case, push servers that reside in IoT nodes can stream data to the collector directly while traffic only passes through the border router.

```
                    +-------------------------------+
                    |           collector           |
                    +---^-----------^------------^--+
                        |           |            |
                        |           |            |
                        |           |            |
                        |   +-------+--------+   |
                        |   |  border router |   |
                        |   +----^------^----+   |
                        |        |      |        |
                        |        |      |        |
                        |    +---+      +---+    |
                        |    |              |    |
                    +---+----+---+      +---+----+---+
                    | IoT node 1 |      | IoT node 2 |
                    +------------+      +------------+
```

Fig. 2 IoT Data Collection

3. Solution Overview

All the use cases described in the previous section are very similar on the data subscription and publication mode, hence can be abstracted to the following generic distributed data collection framework, as shown in the following figure.

A Collector usually includes two components,

- o the Subscriber generates the subscription instructions to express what and how the collector want to receive the data;

- o the Receiver is the target for the data publication.

For one subscription, there may be one to many receivers. And the subscriber does not necessarily share the same address with receivers.

In this framework, the stream originators have the Master role and the Agent role. Both the Master and the Agent include two components,

- o the Subscription Server manages capabilities that it can provide to the subscriber.

- o the Publisher pushes data to the receiver according to the subscription information.

The Master knows all the capabilities that the attached Agents and itself can provide, and exposes the Global Capability to the Collector. The Collector cannot see the Agents directly, so it will only send the Global Subscription information to the Master. The Master disassembles the Global Subscription to multiple Component Subscriptions, each involving data from a separate telemetry source. The Component Subscriptions are then distributed to the corresponding Agents.

When data streaming, the Publisher located in each stream originator collects and encapsulates the packets per the Component Subscription, and pushes the piece of data which can serve directly to the designated data Collector. The Collector is able to assemble many pieces of data associated with one Global Subscription, and can also deduce the missing pieces of data.

```
                +-------------------------------------+
                |      Collector      +-------------+ |
                |                    +|-----------+ | |
                |   +------------+   || Receiver  | <-----+
                |   | Subscriber |   |+-------------+ |   |
                |   +-^----+-----+   +------------+   |   |
                |     |    |                |         |   |
                +-------------------------------------+   |
           Global     |    |Global          | push        |
           Capability |    |Subscription    |             |
                +-------------------------------------+   |
                |     |    |   Master       |         |   |
                |  +--+----v------+  +------+------+  |   |
                |  | Subscription |  | Publisher   |  |   |
                |  | Server       |  |             |  |   |
                |  +--^----+------+  +-------------+  |   |
                |     |    |                          |   |
                +-------------------------------------+   |
           Component  |    | Component               push |
           Capability |    | Subscription                 |
                +-------------------------------------+   |
                |     |    |   Agent                  |   |
                |  +--+----v------+  +-------------+  |   |
                |  | Component    |  | Publisher   |  |   |
                |  | Subscription |  |             +------+
                |  | Server       |  +-------------+  |
                |  +--------------+                   |
                +-------------------------------------+
```

Fig. 3 The Generic Distributed Data Collection Framework

Master and Agents may interact with each other in several ways:

- o Agents need to have a registration or announcement handshake with the Master, so the Master is aware of them and of life-cycle events (such as Agent appearing and disappearing).

- o Contracts are needed between the Master and each Agent on the Component Capability, and the format for streaming data structure.

- o The Master relays the component subscriptions to the Agents.

- o The Agents indicate status of Component Subscriptions to the Master. The status of the overall subscription is maintained by the Master. The Master is also responsible for notifying the subscriber in case of any problems of Component Subscriptions.

Any technical mechanisms or protocols used for the coordination of operational information between Master and Agent is out-of-scope of the solution. We will need to instrument the results of this coordination on the Master Node.

Note: Some preliminary considerations on the solution details are now listed in the appendix for reference. The detailed solution need to be discussed and will be added if the WG accepts the problem statement.

4. Subscription Decomposition

Since Agents are invisible to the Collector, the Collector can only subscribe to the Master. This requires the Master to:

1. expose the Global Capability that can be served by multiple stream originators;

2. disassemble the Global Subscription to multiple Component Subscriptions, and distribute them to the corresponding telemetry sources;

3. notify on changes between portions of a subscription moving between different Agents over time.

To achieve the above requirements, the Master need a Global Capability description which is typically the YANG [RFC7950] data model. This global YANG model is provided as the contract between the Master and the Collector. Each Agent associating with the Master owns a local YANG model to describe the Component Capabilities which it can serve as part of the Global Capability. All the Agents need to know the namespace associated with the Master.

The Master also need a data structure, typically a Resource-Location Table, to keep track of the mapping between the resource and the corresponding location of the Subscription Server which commits to serve the data. When a Global Subscription request arrives, the Master will firstly extract the filter information from the request. Consequently, according to the Resource-Location Table, the Global Subscription can be disassembled into multiple Component Subscriptions, and the corresponding location can be associated.

The decision whether to decompose a Global Subscription into multiple Component Subscriptions rests with the Resource-Location Table. A Master can decide to not decompose a Global Subscription at all and push a single stream to the receiver, because the location information indicates the Global Subscription can be served locally by the Master. Similarly, it can decide to entirely decompose a Global Subscription into multiple Component Subscriptions that each push their own streams, but not from the Master. It can also decide to decompose the Global Subscription into several Component Subscriptions and retain some aspects of the Global Subscription itself, also pushing its own stream.

Component Subscriptions belonging to the same Global Subscription MUST NOT overlap. The combination of all Component Subscriptions MUST cover the same range of nodes as the Global Subscription. Also, the same subscription settings apply to each Component Subscription, i.e., the same receivers, the same time periods, the same encodings are applied to each Component Subscription per the settings of the Global Subscription.

Each Component Subscription in effect constitutes a full-fledged subscription, with the following constraints:

- o Component subscriptions are system-controlled, i.e. managed by the Master Node, not by the subscriber.

- o Component subscription settings such as time periods, dampening periods, encodings, receivers adopt the settings of their Global Subscription.

- o The life-cycle of the Component Subscription is tied to the life-cycle of the Global Subscription. Specifically, terminating/removing the Global Subscription results in termination/removal of Component Subscriptions.

- o The Component Subscriptions share the same Subscription ID as the Global Subscription.

5. Publication Composition

The Publisher collects data and encapsulates the packets per the component subscription. There are several potential encodings, including XML, JSON, CBOR and GPB. The format and structure of the data records are defined by the YANG schema, so that the composition at the Receiver can benefit from the structured and hierarchical data instance.

The Receiver is able to assemble many pieces of data associated with one subscription, and can also deduce the missing pieces of data. The Receiver recognizes data records associated with one subscription according the Subscription ID. Data records generated per one subscription are assigned with the same Subscription ID.

For the time series data stream, records are produced periodically from each stream originator. The message arrival time varies because of the distributed nature of the publication. The Receiver assembles data generated at the same time period based on the recording time consisted in each data record. In this case, time synchronization is required for all the steam originators.

6. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

7. Security Considerations

It’s expected to reuse the existing secure transport layer protocols, such as TLS [RFC5246] and DTLS [RFC6347], to secure the telemetry stream. The Collector cannot access the Agent directly but to negotiate the security parameters with the Master. However the data streams are actually generated by the Agents which are invisible to the Collector. So mechanisms may need to consider when adapting secure transport layer protocols here. the detailed solution is TBD.

8. Acknowledgements

9. References

9.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

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

9.2. Informative References

[I-D.ietf-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-ietf-core-coap-pubsub-04 (work in progress), March 2018.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", draft-ietf-netconf-yang-push-15 (work in progress), February 2018.

Appendix A. Change Log

(To be removed by RFC editor prior to publication)

v01

- o Minor revision on Subscription Decomposition
- o Revised terminologies
- o Removed most implementation related text
- o Place holder of two sections: Subscription Management, and Notifications on Subscription State Changes

v02

- o Revised section 4 and 5. Moved them from apendix to the main text.

Appendix B. Subscription Management

A Global Subscription can be rejected for multiple reasons. Some are related to the Subscription Decomposition and Component Subscription. New error codes are defined to indicate why a datastore subscription attempt has failed. The subscription result with the failure reason is returned as part of the RPC response.

Appendix C. Notifications on Subscription State Changes

Each component subscription maintains its own subscription state and is responsible for sending its own OAM notifications (for example, when the component subscription is suspended or when it can resume).

TBD.

Appendix D. Configured Subscription and Call Home

TBD. Only about the message layer which is transport independent.

Authors' Addresses

Tianran Zhou
Huawei
156 Beiqing Rd., Haidian District
Beijing
China

Email: zhoutianran@huawei.com

Guangying Zheng
Huawei
101 Yu-Hua-Tai Software Road
Nanjing, Jiangsu
China

Email: zhengguangying@huawei.com

Eric Voit
Cisco Systems
United States of America

Email: evoit@cisco.com

Alexander Clemm
Huawei
2330 Central Expressway
Santa Clara, California
United States of America

Email: alexander.clemm@huawei.com

Andy Bierman
YumaWorks
United States of America

Email: andy@yumaworks.com