Abstract

This draft defines a NETCONF server configuration data model and a RESTCONF server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listen on, whether call-home is supported, and associated parameters.

Editorial Note (To be removed by RFC Editor)

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

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:

- draft-ietf-netconf-rfc5539bis
- draft-ietf-netconf-restconf
- draft-ietf-netconf-call-home
- draft-ietf-netmod-snmp-cfg

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

- "VVVV" --> the assigned RFC value for this draft
- "WWWW" --> the assigned RFC value for draft-ietf-netconf-rfc5539bis
The following two Appendix sections are to be removed prior to publication:

- Appendix B. Change Log
- Appendix C. Open Issues

Status of This Memo

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

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

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

This Internet-Draft will expire on June 14, 2015.
Copyright Notice

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

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

Table of Contents

1. Introduction ................................................. 4
   1.1. Terminology ........................................... 4
   1.2. Tree Diagrams ......................................... 4
2. Objectives ................................................... 5
   2.1. Support all NETCONF and RESTCONF transports .......... 5
   2.2. Enable each transport to select which keys to use ... 5
   2.3. Support authenticating NETCONF clients certificates ... 5
   2.4. Support mapping authenticated NETCONF client-certificates
to usernames ................................................. 6
   2.5. Support both Listening for connections and Call Home ... 6
   2.6. For Call Home connections ................................. 6
       2.6.1. Support more than one northbound application .... 6
       2.6.2. Support applications having more than one server .. 6
       2.6.3. Support a reconnection strategy ..................... 6
       2.6.4. Support both persistent and periodic connections .. 7
       2.6.5. Reconnection strategy for periodic connections ... 7
       2.6.6. Keep-алives for persistent connections ............ 7
       2.6.7. Customizations for periodic connections .......... 7
3. The NETCONF Server Configuration Model ..................... 8
   3.1. Overview ................................................. 8
       3.1.1. The "session-options" subtree ..................... 8
       3.1.2. The "listen" subtree ................................ 8
       3.1.3. The "call-home" subtree ............................ 9
       3.1.4. The "ssh" subtree ................................ 11
       3.1.5. The "tls" subtree ................................ 11
   3.2. YANG Module ............................................. 12
4. The RESTCONF Server Configuration Model ..................... 25
   4.1. Overview ................................................. 25
       4.1.1. The "listen" subtree ................................. 25
       4.1.2. The "call-home" subtree ............................. 26
       4.2. YANG Module ........................................... 28
1. Introduction

This draft defines a NETCONF [RFC6241] server configuration data model and a RESTCONF [draft-ietf-netconf-restconf] server configuration data model. These data models enable configuration of the NETCONF and RESTCONF services themselves, including which transports are supported, what ports the servers listens on, whether call-home is supported, and associated parameters.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Tree Diagrams

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

- Brackets "[" and "]" enclose list keys.

- Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Objectives

The primary purpose of the YANG modules defined herein is to enable the configuration of the NETCONF and RESTCONF services on a network element. This scope includes the following objectives:

2.1. Support all NETCONF and RESTCONF transports

The YANG module should support all current NETCONF and RESTCONF transports, namely NETCONF over SSH [RFC6242], NETCONF over TLS [draft-ietf-netconf-rfc5539bis], and RESTCONF over TLS [draft-ietf-netconf-restconf], and to be extensible to support future transports as necessary.

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

2.2. 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.

2.3. Support authenticating NETCONF clients certificates

When a certificate is used to authenticate a NETCONF client, either when using the TLS transport or the SSH transport with X.509 certificates [RFC6187], there is a need to configure the server to know how to authenticate the certificates. The server should be able to do this either by using path-validation to a configured trust anchor or by matching the client-certificate to one previously configured.
2.4. Support mapping authenticated NETCONF client-certificates to user names

Some NETCONF transports (e.g., TLS) need additional support to map authenticated transport-level sessions to a NETCONF username. The NETCONF server model defined herein should define an ability for this mapping to be configured.

2.5. Support both Listening for connections and Call Home

The NETCONF and RESTCONF 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 ([draft-ietf-netconf-call-home]), enabling the server to initiate the connection to the client, for both the NETCONF and RESTCONF protocols. 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.

2.6. For Call Home connections

The following objectives only pertain to call home connections.

2.6.1. Support more than one northbound application

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

2.6.2. Support applications having more than one server

An application managing a device may implement a high-availability strategy employing a multiplicity of active and/or passive servers. Therefore, when it is desired for a device to initiate call home connections, it should be able to connect to any of the application’s servers.

2.6.3. Support a reconnection strategy

Assuming an application has more than one server, then it becomes necessary to configure how a device should reconnect to the application should it lose its connection to the application’s servers. Of primary interest is if the device should start with
first server defined in a user-ordered list of servers or with the last server it was connected to. Secondary settings might specify the frequency of attempts and number of attempts per server. Therefore, a reconnection strategy should be configurable.

### 2.6.4. Support both persistent and periodic connections

Applications may vary greatly on how frequently they need to interact with a device, how responsive interactions with devices need to be, and how many simultaneous connections they can support. Some applications may need a persistent connection to devices to optimize real-time interactions, while others prefer periodic interactions in order to minimize resource requirements. Therefore, when it is necessary for devices to initiate connections, the type of connection desired should be configurable.

### 2.6.5. 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.

### 2.6.6. Keep-aliases 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 application requirements, and therefore keep-alive settings should be configurable on a per-application basis.

### 2.6.7. Customizations for periodic connections

If a periodic connection is desired, it is necessary for the device to know how often it should connect. This delay essentially determines how long the application might have to wait to send data to the device. This setting does not constrain how often the device must wait to send data to the application, as the device should immediately connect to the application whenever it has data to send to it.
A common communication pattern is that one data transmission is many times closely followed by another. For instance, if the device needs to send a notification message, there’s a high probability that it will send another shortly thereafter. Likewise, the application may have a sequence of pending messages to send. Thus, it should be possible for a device to hold a connection open until some amount of time of no data being transmitted as transpired.

3. The NETCONF Server Configuration Model

3.1. Overview

3.1.1. The "session-options" subtree

```
module: ietf-netconf-server
  +--rw netconf-server
     +--rw session-options {session-options}?
       +--rw hello-timeout?   uint32
       +--rw idle-timeout?    uint32
```

The above subtree illustrates how the ietf-netconf-server YANG module enables configuration of NETCONF session options, independent of any transport or connection strategy. A feature statement is used for the server to advertise support for configuring these NETCONF server options. Please see the YANG module (Section 3.2) for a complete description of these configuration knobs.

3.1.2. The "listen" subtree
The above subtree illustrates how the ietf-netconf-server YANG module enables configuration for listening for remote connections, as described in [RFC6242] and [draft-ietf-netconf-call-home]. Feature statements are used to limit both if listening is supported at all as well as for which transports. If listening for connections is supported, then the model enables configuring a list of listening endpoints, each configured with a user-specified name (the key field), the transport to use (i.e. SSH, TLS), and the IP address and port to listen on. The port field is optional, defaulting to the transport-specific port when not configured. Please see the YANG module (Section 3.2) for a complete description of these configuration knobs.

3.1.3. The "call-home" subtree
The above subtree illustrates how the ietf-netconf-server YANG module enables configuration for call home, as described in [draft-ietf-netconf-call-home]. Feature statements are used to limit both if call-home is supported at all as well as for which transports, if it is. If call-home is supported, then the model supports configuring a list of applications to connect to. Each application is configured with a user-specified name (the key field), the transport to be used (i.e. SSH, TLS), and a list of remote...
endpoints, each having a name, an IP address, and an optional port. Additionally, the configuration for each remote application indicates the connection-type (persistent vs. periodic) and associated parameters, as well as the reconnection strategy to use. Please see the YANG module (Section 3.2) for a complete description of these configuration knobs.

3.1.4. The "ssh" subtree

module: ietf-netconf-server
  +--rw netconf-server
    +--rw ssh {ssh}?
      +--rw x509 {rfc6187}?
        +--rw trusted-ca-certs
          | +--rw trusted-ca-cert* binary
        +--rw trusted-client-certs
          +--rw trusted-client-cert* binary

The above subtree illustrates how the ietf-netconf-server YANG module enables some SSH configuration independent of if the NETCONF server is listening or calling home. Specifically, when RFC 6187 is supported, this data model provides an ability to configure how client-certificates are authenticated. Please see the YANG module (Section 3.2) for a complete description of these configuration knobs.

3.1.5. The "tls" subtree

module: ietf-netconf-server
  +--rw netconf-server
    +--rw tls {tls}?
      +--rw client-auth
        +--rw trusted-ca-certs
          | +--rw trusted-ca-cert* binary
        +--rw trusted-client-certs
          | +--rw trusted-client-cert* binary
        +--rw cert-maps
          +--rw cert-to-name* [id]
            +--rw id uint32
            +--rw fingerprint x509c2n:tls-fingerprint
            +--rw map-type identityref
            +--rw name string

The above subtree illustrates how the ietf-netconf-server YANG module enables TLS configuration independent of if the NETCONF server is listening or calling home. Specifically, this data model provides 1) an ability to configure how client-certificates are authenticated and 2) how authenticated client-certificates are mapped to NETCONF user
names. Please see the YANG module (Section 3.2) for a complete
description of these configuration knobs.

3.2. YANG Module

This YANG module imports YANG types from [RFC6991], and
[draft-ietf-netmod-snmp-cfg].

<CODE BEGINS> file "ietf-netconf-server@2014-12-11.yang"

module ietf-netconf-server {

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

import ietf-inet-types { // RFC 6991
    prefix inet;
    revision-date 2013-07-15;
}
import ietf-x509-cert-to-name { // RFC ZZZZ
    prefix x509c2n;
    revision-date 2014-05-06;
}

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

contact
    *WG Web: <http://tools.ietf.org/wg/netconf/>
    *WG List: <mailto:netconf@ietf.org>
    
    WG Chair: Mehmet Ersue
    <mailto:mehmet.ersue@nsn.com>

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

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

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

Copyright (c) 2014 IETF Trust and the persons identified as
authors of the code. All rights reserved."
Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info).

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

revision "2014-12-11" {
  description
    "Initial version";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// Features

feature session-options {
  description
    "The session-options feature indicates that the NETCONF server supports the session-options container.";
}

feature ssh {
  description
    "The ssh feature indicates that the server supports the SSH transport protocol.";
  reference
    "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
}

feature tls {
  description
    "The tls feature indicates that the server supports the TLS transport protocol.";
  reference
    "RFC 5539: NETCONF over Transport Layer Security (TLS)";
}

feature listen {
  description
    "The listen feature indicates that the server supports opening a port to listen for incoming client connections.";
  reference
    "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
feature call-home {
    description
    "The call-home feature indicates that the server supports connecting to the client";
    reference
    "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

feature rfc6187 {
    description
    "The rfc6187 feature indicates that the NETCONF server supports RFC 6187";
    reference
    "RFC 6187: X.509v3 Certificates for Secure Shell Authentication";
}

// top-level container  (groupings below)
container netconf-server {
    description
    "Top-level container for NETCONF server configuration.";

    uses session-options-container;
    uses listen-container;
    uses call-home-container;
    uses ssh-container;
    uses tls-container;
}

grouping session-options-container {
    description
    "";

carier session-options {
    description
    "NETCONF session options, independent of transport or connection strategy.";
    if-feature session-options;
    leaf hello-timeout {
        type uint32 {
            range "0 | 10 .. 3600";
        }
        units "seconds";
}
default '600';
description
"Specifies the number of seconds that a session may exist
before the hello PDU is received. A session will be
dropped if no hello PDU is received before this number
of seconds elapses.

If this parameter is set to zero, then the server will
wait forever for a hello message, and not drop any
sessions stuck in 'hello-wait' state.

Setting this parameter to zero may permit denial of
service attacks, since only a limited number of
concurrent sessions may be supported by the server."
}
leaf idle-timeout {
  type uint32 {
    range "0 | 10 .. 360000";
  }
  units "seconds";
  default '3600';
  description
  "Specifies the number of seconds that a NETCONF session may
remain idle without issuing any RPC requests. A session
will be dropped if it is idle for an interval longer than
this number of seconds. If this parameter is 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.

This mechanism is independent of keep-alives, as it regards
activity occurring at the NETCONF protocol layer, whereas
the keep-alive mechanism regards transport-level activity.";
}
}

grouping listen-container {
  description
  "";
  container listen {
    description
    "Configures listen behavior";
    if-feature listen;
    leaf max-sessions {
      type uint16 {
        range "0 .. 1024";
      }
    }
  }
}
default '0';
description
"Specifies the maximum number of concurrent sessions
that can be active at one time. The value 0 indicates
that no artificial session limit should be used."
}
list endpoint {
  key name;
description
"List of endpoints to listen for connections on."
leaf name {
  type string;
description
"An arbitrary name for the listen endpoint."
}
choice transport {
  mandatory true;
description
"Selects between SSH and TLS transports."
  case ssh {
    if-feature ssh;
    container ssh {
      description
"SSH-specific listening configuration for inbound
connections."
      uses address-and-port-grouping {
        refine port {
          default 830;
        }
      }
      uses host-keys-container;
    }
  }
  case tls {
    if-feature tls;
    container tls {
      description
"TLS-specific listening configuration for inbound
connections."
      uses address-and-port-grouping {
        refine port {
          default 6513;
        }
      }
      uses certificates-container;
    }
  }
}
uses keep-alives-container {
  refine keep-alives/interval-secs {
    default 0; // disabled by default for listen connections
  }
}

grouping call-home-container {
  description "";
  container call-home {
    if-feature call-home;
    description "Configures call-home behavior";
    list application {
      key name;
      description "List of NETCONF clients the NETCONF server is to initiate call-home connections to.";
      leaf name {
        type string;
        description "An arbitrary name for the remote NETCONF client.";
      }
      choice transport {
        mandatory true;
        description "Selects between available transports.";
        case ssh {
          if-feature ssh;
          container ssh {
            description "Specifies SSH-specific call-home transport configuration.";
            uses endpoints-container {
              refine endpoints/endpoint/port {
                default 7777;
              }
            }
            uses host-keys-container;
          }
        }
        case tls {
          if-feature tls;
          container tls {
            description
            "";
          }
        }
      }
    }
  }
}
"Specifies TLS-specific call-home transport configuration.";
uses endpoints-container {
    refine endpoints/endpoint/port {
        default 8888;
    }
}
uses certificates-container;
}
}
container connection-type {
    description "Indicates the kind of connection to be maintained.";
choice connection-type {
    default persistent-connection;
    description "Selects between persistent and periodic connections.";
    case persistent-connection {
        container persistent {
            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.";
uses keep-alives-container {
    refine keep-alives/interval-secs {
        default 15; // 15 seconds for call-home sessions
    }
}
}
}
container periodic {
    description "Periodically connect to NETCONF client, using the reconnection strategy, so the NETCONF client can deliver pending messages to the NETCONF server.

For messages the NETCONF server wants to send to the NETCONF client, the NETCONF server should proactively connect to the NETCONF client, if not already, to send the messages immediately.";
leaf timeout-mins {
}
type uint8;
units minutes;
default 5;
description
"The maximum amount of unconnected time the NETCONF server will wait until establishing a connection to the NETCONF client again. The NETCONF server MAY establish a connection before this time if it has data it needs to send to the NETCONF client. Note: this value differs from the reconnection strategy’s interval-secs value.";
}
leaf linger-secs {
  type uint8;
  units seconds;
  default 30;
  description
  "The amount of time the NETCONF server should wait after last receiving data from or sending data to the NETCONF client’s endpoint before closing its connection to it. This is an optimization to prevent unnecessary connections.";
}
}
}
}

container reconnect-strategy {
  description
  "The reconnection strategy guides how a NETCONF server reconnects to an NETCONF client, after losing a connection to it, even if due to a reboot. The NETCONF server starts with the specified endpoint and tries to connect to it count-max times, waiting interval-secs between each connection attempt, 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. NETCONF servers SHOULD support this flag across reboots.";
      }
    }
  }
}
leaf interval-secs {
  type uint8;
  units seconds;
  default 5;
  description
  "Specifies the time delay between connection attempts to the same endpoint. Note: this value differs from the periodic-connection’s timeout-mins value.";
}
leaf count-max {
  type uint8;
  default 3;
  description
  "Specifies the number times the NETCONF server tries to connect to a specific endpoint before moving on to the next endpoint in the list (round robin).";
}
grouping tls-container {
  description "";
  container tls {
    description "Configures TLS properties not specific to the listen or call-home use-cases";
    if-feature tls;
    container client-auth {
      description "Container for TLS client authentication configuration.";
      uses trusted-certs-grouping;
      container cert-maps {
        uses x509c2n:cert-to-name;
        description "The cert-maps container is used by a 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.";
      }
    }
  }
}


grouping trusted-certs-grouping {
  description "";
  container trusted-ca-certs {
    description "A list of Certificate Authority (CA) certificates that a NETCONF server can use to authenticate NETCONF client certificates. A client’s certificate is authenticated if there is a chain of trust to a configured trusted CA certificate. The client certificate MAY be accompanied with additional certificates forming a chain of trust. The client’s certificate is authenticated if there is path-validation from any of the certificates it presents to a configured trust anchor.";
    leaf-list trusted-ca-cert {
      type binary;
      ordered-by system;
      description "The binary certificate structure as specified by RFC 5246, Section 7.4.6, i.e.,: opaque ASN.1Cert<1..2^24>;
      ";
      reference
    }
  }
}

};

} container trusted-client-certs {
  description
  "A list of client certificates that a NETCONF server can use to authenticate a NETCONF client’s certificate. A client’s certificate is authenticated if it is an exact match to a configured trusted client certificates."
  leaf-list trusted-client-cert {
    type binary;
    ordered-by system;
    description
      "The binary certificate structure, as specified by RFC 5246, Section 7.4.6, i.e.,:

          opaque ASN.1Cert<1..2^24>;

      ";
    reference
  }
}

grouping host-keys-container {
  description
  "";
  container host-keys {
    description
      "Parent container for the list of host-keys."
    leaf-list host-key {
      type string;
      min-elements 1;
      ordered-by user;
      description
        "A user-ordered list of host-keys the SSH server considers when composing the list of server host key algorithms it will send to the client in its SSH_MSG_KEXINIT message. The value of the string is the unique identifier for a host-key configured on the system. How valid values are discovered is outside the scope of this module, but they are envisioned to be the keys for a list of host-keys provided by another YANG module";
      reference
  }
}
grouping certificates-container {
    description "";
    container certificates {
        description "Parent container for the list of certificates.";
        leaf-list certificate {
            type string;
            min-elements 1;
            description "An unordered list of certificates the TLS server can pick from when sending its Server Certificate message. The value of the string is the unique identifier for a certificate configured on the system. How valid values are discovered is outside the scope of this module, but they are envisioned to be the keys for a list of certificates provided by another YANG module";
            reference "RFC 5246: The TLS Protocol, Section 7.4.2";
        }
    }
}

grouping address-and-port-grouping {
    description "a common grouping";
    leaf address {
        type inet:ip-address;
        description "The IP address of the interface to listen on.";
    }
    leaf port {
        type inet:port-number;
        description "The local port number on this interface the NETCONF server listens on.";
    }
}

grouping endpoints-container {
    description "Grouping for transport-specific configuration for
call-home connections; 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 NETCONF client. Defining more than one enables high-availability.";
    leaf name {
      type string;
      description "An arbitrary name for the endpoint to connect to.";
    }
    leaf address {
      type inet:host;
      mandatory true;
      description "The hostname or IP address or hostname of the endpoint. If a hostname is provided and DNS resolves to more than one IP address, the NETCONF server SHOULD try all of the ones it can based on how its networking stack is configured (e.g. v4, v6, dual-stack).";
    }
    leaf port {
      type inet:port-number;
      description "The IP port for this endpoint. The NETCONF server will use the IANA-assigned well-known port if not specified.";
    }
  }
}

grouping keep-alives-container {
  description "";
  container keep-alives {
    description "Configures the keep-alive policy, to proactively test the aliveness of the NETCONF client, in order to know when a new call home connection should be established.";
    reference "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models, Section 4";
    leaf interval-secs {
type uint8;
units seconds;
description
"Sets a timeout interval in seconds after which if no data
has been received from the NETCONF client, a message will
be sent to request a response from the NETCONF client. A
value of '0' indicates that no keep-alive messages should
be sent.";
}
leaf count-max {
  type uint8;
  default 3;
  description
  "Sets the number of keep-alive messages that may be sent
  without receiving any data from the NETCONF client before
  assuming the NETCONF client is no longer alive. If this
  threshold is reached, the transport-level connection will
  be disconnected, which will trigger the reconnection
  strategy). The interval timer is reset after each
  transmission, thus an unresponsive NETCONF client will
  be dropped after ~count-max * interval-secs seconds.";
}
}

<CODE ENDS>

4. The RESTCONF Server Configuration Model

4.1. Overview

4.1.1. The "listen" subtree
The above subtree illustrates how the ietf-restconf-server YANG module enables configuration for listening for remote connections, as described in [draft-ietf-netconf-restconf] and [draft-ietf-netconf-call-home]. Feature statements are used to limit both if listening is supported at all as well as for which transports. If listening for connections is supported, then the model enables configuring a list of listening endpoints, each configured with a user-specified name (the key field), the transport to use (i.e. SSH, TLS), and the IP address and port to listen on. The port field is optional, defaulting to the transport-specific port when not configured. Please see the YANG module (Section 4.2) for a complete description of these configuration knobs.

4.1.2. The "call-home" subtree
The above subtree illustrates how the ietf-restconf-server YANG module enables configuration for call home, as described in [draft-ietf-netconf-call-home]. Feature statements are used to limit both if call-home is supported at all as well as for which transports, if it is. If call-home is supported, then the model supports configuring a list of applications to connect to. Each application is configured with a user-specified name (the key field), the transport to be used (i.e. SSH, TLS), and a list of remote endpoints, each having a name, an IP address, and an optional port. Additionally, the configuration for each remote application indicates the connection-type (persistent vs. periodic) and associated parameters, as well as the reconnection strategy to use. Please see the YANG module (Section 4.2) for a complete description of these configuration knobs.
4.2. YANG Module

This YANG module imports YANG types from [RFC6991].

<CODE BEGINS> file "ietf-restconf-server@2014-12-11.yang"

module ietf-restconf-server {
    namespace "urn:ietf:params:xml:ns:yang:ietf-restconf-server";
    prefix "rcserver";

    import ietf-inet-types {           // RFC 6991
        prefix inet;
        revision-date 2013-07-15;
    }

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

    contact
        "WG Web: <http://tools.ietf.org/wg/netconf/>
        WG List: <mailto:netconf@ietf.org>
        WG Chair: Mehmet Ersue
            <mailto:mehmet.ersue@nsn.com>
        WG Chair: Mahesh Jethanandani
            <mailto:mjethanandani@gmail.com>
        Editor: Kent Watsen
            <mailto:kwatsen@juniper.net>
    ";

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

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

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

        This version of this YANG module is part of RFC VVVV; see

        ";

Watsen & Schoenwaelder Expires June 14, 2015 [Page 28]
the RFC itself for full legal notices.

revision "2014-12-11" {
  description
  "Initial version";
  reference
  "RFC VVVV: NETCONF Server and RESTCONF Server Configuration Models";
}

// Features

feature tls {
  description
  "The tls feature indicates that the server supports RESTCONF
   over the TLS transport protocol.";
  reference
  "RFC XXXX: RESTCONF Protocol";
}

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

feature call-home {
  description
  "The call-home feature indicates that the server supports
   connecting to the client";
  reference
  "RFC YYYY: NETCONF Call Home and RESTCONF Call Home";
}

// top-level container (groupings below)
container restconf-server {
  description
  "Top-level container for RESTCONF server configuration.";

  uses listen-container;
  uses call-home-container;
}

grouping listen-container {

}
description
  
container listen {
  
  description
    "Configures listen behavior";
  if-feature listen;
  leaf max-sessions {
    type uint16 {
      range "0 .. 1024";
    }
    default '0';
    description
      "Specifies the maximum number of concurrent sessions
         that can be active at one time. The value 0 indicates
         that no artificial session limit should be used.";
  }
  list endpoint {
    key name;
    description
      "List of endpoints to listen for connections on.";
    leaf name {
      type string;
      description
        "An arbitrary name for the listen endpoint.";
    }
    choice transport {
      mandatory true;
      description
        "Selects between available transports.";
      case tls {
        container tls {
          description
            "TLS-specific listening configuration for inbound
               connections.";
          uses address-and-port-grouping {
            refine port {
              default 6513;
            }
          }
          uses certificates-container;
        }
      }
      uses keep-alives-container {
        refine keep-alives/interval-secs {
          default 0; // disabled by default for listen connections
        }
      }
    }
  }
}
grouping call-home-container {
    description "";
    container call-home {
        if-feature call-home;
        description "Configures call-home behavior";
        list application {
            key name;
            description "List of RESTCONF clients the RESTCONF server is to initiate call-home connections to.";
            leaf name {
                type string;
                description "An arbitrary name for the remote RESTCONF client.";
            }
            choice transport {
                mandatory true;
                description "Selects between SSH and TLS transports.";
                case tls {
                    if-feature tls;
                    container tls {
                        description "Specifies TLS-specific call-home transport configuration.";
                        uses endpoints-container {
                            refine endpoints/endpoint/port {
                                default 9999;
                            }
                        }
                        uses certificates-container;
                    }
                }
            }
            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 persistent and periodic connections.";
                }
            }
        }
    }
}

case persistent-connection {
    container persistent {
        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.";
        uses keep-alives-container {
            refine keep-alives/interval-secs {
                default 15; // 15 seconds for call-home sessions
            }
        }
    }
}

case periodic-connection {
    container periodic {
        description
        "Periodically connect to RESTCONF client, using the reconnection strategy, so the RESTCONF client can deliver pending messages to the RESTCONF server.

        For messages the RESTCONF server wants to send to the RESTCONF client, the RESTCONF server should proactively connect to the RESTCONF client, if not already, to send the messages immediately.";
        leaf timeout-mins {
            type uint8;
            units minutes;
            default 5;
            description
            "The maximum amount of unconnected time the RESTCONF server will wait until establishing a connection to the RESTCONF client again. The RESTCONF server MAY establish a connection before this time if it has data it needs to send to the RESTCONF client. Note: this value differs from the reconnection strategy’s interval-secs value.";
        }
        leaf linger-secs {
            type uint8;
            units seconds;
            default 30;
            description
            "The amount of time the RESTCONF server should wait
after last receiving data from or sending data to
the RESTCONF client’s endpoint before closing its
connection to it. This is an optimization to
prevent unnecessary connections.

The reconnection strategy guides how a RESTCONF server
reconnects to an RESTCONF client, after losing a connection
to it, even if due to a reboot. The RESTCONF server starts
with the specified endpoint and tries to connect to it
count-max times, waiting interval-secs between each
connection attempt, 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. RESTCONF servers
      SHOULD support this flag 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. If no previous connection has
ever been established, last-connected defaults to
the first endpoint listed.";
}

'leaf interval-secs {
  type uint8;
  units seconds;
  default seconds 5;
  description
  "Specifies the time delay between connection attempts
to the same endpoint. Note: this value differs from
the periodic-connection’s timeout-mins value.";

leaf count-max {
  type uint8;
  default 3;
  description
  "Specifies the number times the RESTCONF server tries to
  connect to a specific endpoint before moving on to the
  next endpoint in the list (round robin).";
}

grouping certificates-container {
  description
  "";
  container certificates {
    description
    "Parent container for the list of certificates.";
    leaf-list certificate {
      type string;
      min-elements 1;
      description
      "An unordered list of certificates the TLS server can pick
      from when sending its Server Certificate message. The value
      of the string is the unique identifier for a certificate
      configured on the system. How valid values are discovered
      is outside the scope of this module, but they are envisioned
      to be the keys for a list of certificates provided
      by another YANG module";
      reference
      "RFC 5246: The TLS Protocol, Section 7.4.2";
    }
  }
}

grouping address-and-port-grouping {
  description
  "a common grouping";
  leaf address {
    type inet:ip-address;
    description
    "The IP address of the interface to listen on.";
  }
  leaf port {
    type inet:port-number;
    description
  }
}
"The local port number on this interface the RESTCONF server listens on."
}
}

grouping endpoints-container {
    description
        "Grouping for transport-specific configuration for call-home connections."
    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 the endpoint to connect to."
            }
            leaf address {
                type inet:host;
                mandatory true;
                description
                    "The hostname or IP address or hostname of the endpoint. If a hostname is provided and DNS resolves to more than one IP address, the RESTCONF server SHOULD try all of the ones it can based on how its networking stack is configured (e.g. v4, v6, dual-stack)."
            }
            leaf port {
                type inet:port-number;
                description
                    "The IP port for this endpoint. The RESTCONF server will use the IANA-assigned well-known port if not specified."
            }
        }
    }
}

grouping keep-alives-container {
    description
        ""
container keep-alives {
  description
    "Configures the keep-alive policy, to proactively test the
    aliveness of the RESTCONF client, in order to know when a
    new call home connection should be established.";
  reference
    "RFC VVVV: NETCONF Server and RESTCONF Server Configuration
     Models, Section 4";
  leaf interval-secs {
    type uint8;
    units seconds;
    description
      "Sets a timeout interval in seconds after which if no data
      has been received from the RESTCONF client, a message will
      be sent to request a response from the RESTCONF client. A
      value of '0' indicates that no keep-alive messages should
      be sent.";
  }
  leaf count-max {
    type uint8;
    default 3;
    description
      "Sets the number of keep-alive messages that may be sent
      without receiving any data from the RESTCONF client before
      assuming the RESTCONF client is no longer alive. If this
      threshold is reached, the transport-level connection will
      be disconnected, which will trigger the reconnection
      strategy). The interval timer is reset after each
      transmission, thus an unresponsive RESTCONF client will
      be dropped after ~count-max * interval-secs seconds.";
    }
  }
}<CODE ENDS>

5. Implementation strategy for keep-alives

One of the objectives listed above, Keep-alives for persistent
connections Section 2.6.6, indicates a need for a "keep-alive"
mechanism. This section specifies how the keep-alive mechanism is to
be implemented for both the SSH and TLS transports.

Both SSH and TLS have the ability to support keep-alives securely.
Using the strategies listed below, the keep-alive messages are sent
inside the encrypted tunnel and thus immune to attack.
5.1. Keep-alives for SSH

The SSH keep-alive solution that is expected to be used is ubiquitous in practice, though never being explicitly defined in an RFC. The strategy used is to purposely send a malformed request message with a flag set to ensure a response. More specifically, per section 4 of [RFC4253], either SSH peer can send a SSH_MSG_GLOBAL_REQUEST message with "want reply" set to '1' and that, if there is an error, will get back a SSH_MSG_REQUEST_FAILURE response. Similarly, section 5 of [RFC4253] says that either SSH peer can send a SSH_MSG_CHANNEL_REQUEST message with "want reply" set to '1' and that, if there is an error, will get back a SSH_MSG_CHANNEL_FAILURE response.

To ensure that the request will fail, current implementations of this keep-alive strategy (e.g. OpenSSH’s 'sshd' server) send an invalid "request name" or "request type", respectively. Abiding to the extensibility guidelines specified in Section 6 of [RFC4251], these implementations use the "name@domain". For instance, when configured to send keep-alives, OpenSSH sends the string "keepalive@openssh.com". In order to remain compatible with existing implementations, this draft does not require a specific "request name" or "request type" string be used, implementations are free to pick values of their choosing.

5.2. Keep-alives for TLS

The TLS keep-alive solution that is expected to be used is defined in [RFC6520]. This solution allows both peers to advertise if they can receive heartbeat request messages from its peer. For standard TLS connections, devices SHOULD advertise "peer_allowed_to_send", as per [RFC6520]. This advertisement is not a "MUST" in order to grandfather existing NETCONF/RESTCONF over TLS implementations. For NETCONF Call Home or RESTCONF Call Home, the network management system MUST advertise "peer_allowed_to_send" per [RFC6520]. This is a "MUST" so as to ensure devices can depend on it always being there for call home connections, which is when keep-alives are needed the most.

6. Security Considerations

The YANG modules defined in this memo are designed to be accessed via the NETCONF protocol [RFC6241]. Authorization for access to specific portions of conceptual data and operations within this module is provided by the NETCONF access control model (NACM) [RFC6536].

There are a number of data nodes defined in the "ietf-netconf-server" YANG module which are readable and/or writable that may be considered
sensitive or vulnerable in some network environments. Write and read operations to these data nodes can have a negative effect on network operations. It is thus important to control write and read access to these data nodes. Below are the data nodes and their sensitivity/vulnerability.

netconf-server/tls/client-auth/trusted-ca-certs:

- This container contains certificates that the server is to use as trust anchors for authenticating TLS-specific client certificates. Write access to this node should be protected.

netconf-server/tls/client-auth/trusted-client-certs:

- This container contains certificates that the server is to trust directly when authenticating TLS-specific client certificates. Write access to this node should be protected.

netconf-server/tls/client-auth/cert-map:

- This container contains a user name that some deployments may consider sensitive information. Read access to this node may need to be guarded.

7. IANA Considerations

This document registers two URIs in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registrations are requested:

```
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

```
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:
8. Other Considerations

The YANG modules define herein do not themselves support virtual routing and forwarding (VRF). It is expected that external modules will augment in VRF designations when needed.

9. 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, David Lamparter, Alan Luchuk, Ladislav Lhotka, Radek Krejci, Tom Petch, Phil Shafer, and Bert Wijnen.

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

10. References

10.1. Normative References


10.2. Informative References

Appendix A. Examples

A.1. NETCONF Configuration using SSH Transport

The following example illustrates the <get> response from a NETCONF server that only supports SSH, both listening for incoming connections as well as calling home to a single application having two endpoints.

```xml
<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <session-options>
    <hello-timeout>600</hello-timeout>
    <idle-timeout>3600</idle-timeout>
  </session-options>
  <listen>
    <endpoint>
      <name>foo bar</name>
      <ssh>
        <address>11.22.33.44</address>
        <host-keys>
          <host-key>my-rsa-key</host-key>
          <host-key>my-dss-key</host-key>
        </host-keys>
      </ssh>
    </endpoint>
  </listen>
  <call-home>
    <application>
      <name>config-mgr</name>
      <ssh>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <address>11.22.33.44</address>
          </endpoint>
          <endpoint>
            <name>west-data-center</name>
            <address>55.66.77.88</address>
          </endpoint>
        </endpoints>
        <host-keys>
          <host-key>my-call-home-x509-key</host-key>
        </host-keys>
      </ssh>
    </application>
  </call-home>
</netconf-server>
```
<trusted-ca-certs>
<trusted-ca-cert>
    QW4gRWzdGVyIGVnZywgZm9yIHRob3NlIHDobyBtaWdodCBsb29rICA6KQo=
</trusted-ca-cert>
</trusted-ca-certs>
<trusted-client-certs>
<trusted-client-cert>
    SSBhbSB0aGUgZWdnIG1hbiwgdGhleSBhcml2YXRlIGl1bGx5Ig==
</trusted-client-cert>
<trusted-client-cert>
    SSBhbSB0aGUgd2FscnVzLCBnb28gZ29vIGcnam9vYi4K
</trusted-client-cert>
</trusted-client-certs>
</netconf-server>

A.2. NETCONF Configuration using TLS Transport

The following example illustrates the <get> response from a NETCONF server that only supports TLS, both listening for incoming connections as well as calling home to a single application having two endpoints. Please note also the configurations for authenticating client certificates and mappings authenticated certificates to NETCONF user names.

<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
    <session-options>
        <hello-timeout>600</hello-timeout>
        <idle-timeout>3600</idle-timeout>
    </session-options>
    <listen>
        <endpoint>
            <name>primary-netconf-endpoint</name>
            <tls>
                <address>11.22.33.44</address>
                <certificates>
                    <certificate>fw1.east.example.com</certificate>
                </certificates>
            </tls>
        </endpoint>
    </listen>
    <call-home>
        <application>
            <name>config-mgr</name>
        </application>
    </call-home>
</netconf-server>
A.3. RESTCONF Configuration using TLS Transport

The following example illustrates the <get> response from a RESTCONF server that only supports TLS, both listening for incoming connections as well as calling home to a single application having two endpoints.

```
<restconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf-server">
  <listen>
    <endpoint>
      <name>primary-restconf-endpoint</name>
      <tls>
        <address>11.22.33.44</address>
        <certificates>
          <certificate>fw1.east.example.com</certificate>
        </certificates>
      </tls>
    </endpoint>
  </listen>
  <call-home>
    <application>
      <name>config-mgr</name>
      <tls>
        <endpoints>
          <endpoint>
            <name>east-data-center</name>
            <address>11.22.33.44</address>
          </endpoint>
          <endpoint>
            <name>west-data-center</name>
            <address>55.66.77.88</address>
          </endpoint>
        </endpoints>
        <certificates>
          <certificate>fw1.east.example.com</certificate>
        </certificates>
      </tls>
    </application>
  </call-home>
</restconf-server>
```

Appendix B. Change Log

B.1. 00 to 01

- Restructured document so it flows better
- Added trusted-ca-certs and trusted-client-certs objects into the
  ietf-system-tls-auth module

B.2. 01 to 02
- removed the "one-to-many" construct
- removed "address" as a key field
- removed "network-manager" terminology
- moved open issues to github issues
- brought TLS client auth back into model

B.3. 02 to 03
- fixed tree diagrams and surrounding text

B.4. 03 to 04
- reduced the number of grouping statements
- removed psk-maps and associated feature statements
- added ability for listen/call-home instances to specify which
  host-keys/certificates (of all listed) to use
- clarified that last-connected should span reboots
- added missing "objectives" for selecting which keys to use,
  authenticating client-certificates, and mapping authenticated
  client-certificates to usernames
- clarified indirect client certificate authentication
- added keep-alive configuration for listen connections
- added global-level NETCONF session parameters

B.5. 04 to 05
- Removed all refs to the old ietf-system-tls-auth module
- Removed YANG 1.1 style if-feature statements (loss some
  expressiveness)
- Removed the read-only (config false) lists of SSH host-keys and TLS certs
- Added an if-feature around session-options container
- Added ability to configure trust-anchors for SSH X.509 client certs
- Now imports by revision, per best practice
- Added support for RESTCONF server
- Added RFC Editor instructions

Appendix C. Open Issues

Please see: https://github.com/netconf-wg/server-model/issues.

Authors’ Addresses

Kent Watsen
Juniper Networks
EMail: kwatsen@juniper.net

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