NETCONF Server Configuration Model
draft-ietf-netconf-server-model-04

Abstract

This draft defines a NETCONF server configuration data model. This data model enables configuration of the NETCONF service itself, including which transports it supports, what ports they listen on, whether call-home is supported, and associated parameters.

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 April 29, 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 .................................. 3
   1.1. Terminology .................................. 3
   1.2. Tree Diagrams .................................. 3

2. Objectives ...................................... 3
   2.1. Support all NETCONF transports .............. 3
   2.2. Enable each transport to select which keys to use ... 4
   2.3. Support authenticating client-certificates ....... 4
   2.4. Support mapping authenticated client-certificates to usernames .................................. 4
   2.5. Support both Listening for connections and Call Home ... 4
   2.6. For Call Home connections ...................... 4
       2.6.1. Support more than one application ......... 4
       2.6.2. Support applications having more than one server .. 5
       2.6.3. Support a reconnection strategy .......... 5
       2.6.4. Support both persistent and periodic connections .. 5
       2.6.5. Reconnection strategy for periodic connections ... 5
       2.6.6. Keep-alives for persistent connections .......... 5
       2.6.7. Customizations for periodic connections .......... 6

3. Data Model ..................................... 6
   3.1. Overview ..................................... 6
       3.1.1. The "session-options" subtree ............. 6
       3.1.2. The "listen" subtree ...................... 6
       3.1.3. The "call-home" subtree ................... 7
       3.1.4. The "ssh" subtree ......................... 9
       3.1.5. The "tls" subtree ......................... 9
   3.2. YANG Module .................................. 10

4. Implementation strategy for keep-alives ................ 24
   4.1. Keep-alives for SSH .......................... 24
   4.2. Keep-alives for TLS .......................... 25

5. Security Considerations .......................... 25

6. IANA Considerations ............................. 26

7. Other Considerations ............................. 26

8. Acknowledgements ............................... 26

9. References ...................................... 27
   9.1. Normative References .......................... 27
   9.2. Informative References ....................... 28

Appendix A. Examples ............................... 29
   A.1. SSH Transport Configuration + State ............. 29
   A.2. TLS Transport Configuration + State ............. 31

Appendix B. Change Log ............................. 32
   B.1. 00 to 01 ..................................... 33
   B.2. 01 to 02 ..................................... 33
   B.3. 02 to 03 ..................................... 33
   B.4. 03 to 04 ..................................... 33

Appendix C. Open Issues ............................. 34
1. Introduction

This draft defines a NETCONF [RFC6241] server configuration data model. This data model enables configuration of the NETCONF service itself, including which transports are supported, what ports the server 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 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 (":").

2. Objectives

The primary purpose of the YANG module defined herein is to enable the configuration of the NETCONF server service on the device. This scope includes the following objectives:

2.1. Support all NETCONF transports

The YANG module should support all current NETCONF transports, namely NETCONF over SSH [RFC6242] and NETCONF over TLS [rfc5539bis], and 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

Systems may have a multiplicity of host-keys or server-certificates from which subsets are configured for specific uses. For instance, a system 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.

2.3. Support authenticating client-certificates

When certificates are used to authenticate NETCONF clients, there is a need to configure the system to know how to authenticate the certificates. The system 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 client-certificates to usernames

Some 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

NETCONF has always supported 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]). The module should configure both listening for connections and call-home.

Because implementations may not support both listening for connections and call 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 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 for 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 are satisfied with periodic interactions and reduced resources required. Therefore, when it is necessary for devices to initiate connections, the type of connection desired should be configured.

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-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
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. Data Model

#### 3.1. Overview

#### 3.1.1. The "session-options" subtree

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

The above subtree illustrates how this YANG module enables configuration of NETCONF session options, independent of any transport or connection strategy. 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 this YANG module enables configuration for listening for remote connections, as described in [RFC6242] and [rfc5539bis]. 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.

3.1.3. The "call-home" subtree
The above subtree illustrates how this 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.

3.1.4. The "ssh" subtree

```
module: ietf-netconf-server
   +--rw netconf-server
      +--rw ssh
         +--ro host-keys
            +--ro host-key* [name]
               +--ro name                 string
               +--ro format-identifier    string
               +--ro data                 binary
               +--ro fingerprint          string
```

The above subtree illustrates how this YANG module provides SSH state independent of if the NETCONF server if listening or calling home. This data-model provides a read-only listing of currently configured TLC certificates.

3.1.5. The "tls" subtree

```
module: ietf-netconf-server
   +--rw netconf-server
      +--rw tls
         +--ro certificates
            |  +--ro certificate* [name]
            |     +--ro name    string
            |     +--ro data    binary
         +--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 this YANG module provides TLS state and enables TLS configuration independent of if the NETCONF server if listening or calling home. This data-model provides 1) a read-only listing of currently configured TLC certificates and 2) an ability to
configure how client-certificates are authenticated and how authenticated client-certificates are mapped to NETCONF user names.

3.2. YANG Module

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

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-netconf-server@2014-10-26.yang"

module ietf-netconf-server {

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

  import ietf-inet-types {
    prefix inet;                // RFC 6991
  }

  import ietf-x509-cert-to-name {
    prefix x509c2n;             // draft-ietf-netmod-snmp-cfg
  }

  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: Bert Wijnen
    <mailto:bertietf@bwijnen.net>

    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 XXXX; see the RFC itself for full legal notices.";
// RFC Ed.: replace XXXX with actual RFC number and
// remove this note

// RFC Ed.: please update the date to the date of publication

revision "2014-10-26" {  // YYYY-MM-DD
description
  "Initial version";
reference
  "RFC XXXX: NETCONF Server Configuration Model";
}

// Features

feature ssh-listen {
description
  "The ssh-listen feature indicates that the NETCONF server can
  open a port to listen for incoming client connections.";
}

feature ssh-call-home {
description
  "The ssh-call-home feature indicates that the NETCONF server can
  connect to a client.";
reference
  "RFC XXXX: Reverse Secure Shell (Reverse SSH)";
}

feature tls-listen {
description
  "The tls-listen feature indicates that the NETCONF server can
  open a port to listen for incoming client connections.";
}

feature tls-call-home {
description
  "The tls-call-home feature indicates that the NETCONF server can
  connect to a client.";
// 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 "";
    container session-options {
        description
            "NETCONF session options, independent of transport or connection strategy.";
        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 are supported by the server.";
        }
        leaf idle-timeout {
            type uint32 {
                range "0 .. 3600";
            }
            units "seconds";
            default '120';
            description
                "Specifies the number of seconds after which a session will be closed if no activity occurs. Setting this parameter to zero will disable session idle timeout. If this parameter is set to a non-zero value, the server will close any inactive session before this number of seconds elapses. The value '0' allows session idle timeout to be disabled. The server will automatically drop any session that remains idle for longer than the keepalive timeout before this number of seconds elapses.

                Setting this parameter to zero may permit denial of service attacks, since only a limited number of concurrent sessions are supported by the server."
        }
    }
}
range "0 | 10 .. 360000";
)
units "seconds";
default '3600';
description
"Specifies the number of seconds that a 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.

Sessions that have a notification subscription
active are never dropped.

If this parameter is set to zero, then the server
will never drop a session because it is idle.";
}
}
}
grouping listen-container {
  description "";
  container listen {
    description "Configures listen behavior";
    //if-feature ",(ssh-listen or tls-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-listen;
       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-listen;
       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 "(ssh-call-home or tls-call-home)"
       description
           "Configures call-home behavior";
       list application {
           key name;
description
"List of applications to call-home to."
leaf name {
  type string;
  description
  "An arbitrary name for the remote application."
}
choice transport {
  mandatory true;
  description
  "Selects between SSH and TLS transports."
  case ssh {
    if-feature ssh-call-home;
    container ssh {
      description
      "Specifies SSH-specific call-home transport configuration."
      uses endpoints-container {
        refine endpoints/endpoint/port {
          default 8888;  // pending IANA assignment
        }
      }
      uses host-keys-container;
    }
  }
  case tls {
    if-feature tls-call-home;
    container tls {
      description
      "Specifies TLS-specific call-home transport configuration."
      uses endpoints-container {
        refine endpoints/endpoint/port {
          default 9999;  // pending IANA assignment
        }
      }
      uses certificates-container;
    }
  }
}
container connection-type {
  description
  "Indicates the NETCONF client’s preference for how the device’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 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
            }
        }
    }
}
case periodic-connection {
    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 device will wait until establishing a connection to the NETCONF client again. The device 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 device 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 device reconnects to an application, after losing a connection to it, even if due to a reboot. The device starts with the specified endpoint, 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 device tries to
  connect to a specific endpoint before moving on to
  the next endpoint in the list (round robin).";
}

grouping ssh-container {
  description
  "";
  container ssh {
    description
    "Configures SSH properties not specific to the listen
    or call-home use-cases";
    //if-feature "(ssh-listen or ssh-call-home)"
    container host-keys {
      config false;
      description
      "Parent container for a list of host keys";
      list host-key {
        key name;
        description
        "A read-only list of host-keys supported by server";
        leaf name {
          type string;
          description
          "Common name for the host-key";
        }
        leaf format-identifier {
          type string;
          mandatory true;
          description
          "ssh-dss, ssh-rsa, x509v3-rsa2048-sha256, etc.";
          reference
          "RFC 4253: SSH Transport Layer Protocol, section 6.6
          RFC 6187: X.509v3 Certificates for SSH, section 3";
        }
        leaf data {

grouping tls-container {
    description "";
    container tls {
        description "Configures TLS properties not specific to the listen or call-home use-cases";
        //if-feature "(tls-listen or tls-call-home)"
        container certificates {
            config false;
            description "Parent container for a list of certificates"
            list certificate {
                key name;
                description "A list of certificates"
                leaf name {
                    type string;
                    description "the certificate’s common name"
                }
                leaf data {
                    type binary;
                    mandatory true;
                    description "The binary certificate structure, as specified"
by RFC 5246, Section 7.4.2, i.e.,: opaque
ASN.1Cert<1..2^24-1>;
}
}
}
container client-auth {
  description
  "Container for TLS client authentication configuration."
  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. Note, in the TLS protocol, 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>;
      ";
    }
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 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 "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. The value of the string is the name of a host-key configured on the system, as returned by /netconf-server/ssh/host-keys/host-key/name.";
      reference "RFC 4253: The SSH Transport Layer Protocol, Section 7";
    }
  }
}

grouping certificates-container {
  description "";
  container certificates {

description
"Parent container for the list of certificates.";
leaf-list certificate {
  type string;
  min-elements 1;
  description
  "Unordered list of certificates the TLS server can
  pick from when sending its Server Certificate
  message. The value of the string is the name of a
  certificate configured on the system, as returned by
  /netconf-server/tls/certificates/certificate/name";
  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 application.
      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 device 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 device 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. Keepalive implementation is described in RFC XXXX, section 4.";
        reference "RFC XXXX: NETCONF Server Configuration Model 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. 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 NETCONF 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 transport sessions.

4.1. 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.

4.2. 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 NETCONF over TLS connections, devices SHOULD advertise "peer_allowed_to_send", as per [RFC6520]. This advertisement is not a "MUST" in order to grandfather existing NETCONF over TLS implementations. For NETCONF 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 in it always being there for call home connections, which is when keep-alives are needed the most.

5. 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 system 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:
o This container contains certificates that the system 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:

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

6. IANA Considerations

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

<table>
<thead>
<tr>
<th>URI</th>
<th>Registrant Contact</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn:ietf:params:xml:ns:yang:ietf-netconf-server</td>
<td>The NETCONF WG of the IETF.</td>
<td>N/A, the requested URI is an XML namespace.</td>
</tr>
<tr>
<td>urn:ietf:params:xml:ns:yang:ietf-system-tls-auth</td>
<td>The NETCONF WG of the IETF.</td>
<td>N/A, the requested URI is an XML namespace.</td>
</tr>
</tbody>
</table>

This document registers two YANG modules in the YANG Module Names registry [RFC6020].

<table>
<thead>
<tr>
<th>name</th>
<th>namespace</th>
<th>prefix</th>
<th>reference</th>
</tr>
</thead>
</table>

7. Other Considerations

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

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

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.

9. References

9.1. Normative References


9.2. Informative References

Appendix A.  Examples

A.1.  SSH Transport Configuration + State

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. Please also note that the list of host-keys at the end is read-only operational state.

```xml
<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <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>
  <ssh>
    <host-keys>
      <host-key>
        <name>my-rsa-key</name>
        <format-identifier>ssh-rsa</format-identifier>
      </host-key>
    </host-keys>
  </ssh>
</netconf-server>
```
<data>  <!-- base64 reformatted for draft -->
AAAAAB3NzaC1yc2EAAABABIwAAAQEA7D21xYg3+WD97RZqZt08bUU8qPl6e9g
X1l2ZHZ8N95Sr+x2H11HCDSejmx/6JiOuiK5eBvbJE9FVF3phs162fupN6
Y4EmXosc6iqpuI4ldcG963XCQ10enWG4ppdqf8t4c58mEClw7MKPzBHK6
rNQc1cmVvUvLFOvBwBu/54QAIiUwvHkAsk8bkN9Yx7JE1NTV1FFQmvMADVcD
2qqPangETwV5iIN8AE8kbYcM/mmHucGNS8axXR3V9R5KqXP2DYGB7d2
k61OnGa3LBoYl/5q+08F1US+ktytfgwuFGUC+Mx7aKReSIAfov3oWVjeBL
KwsvjD24U068qtwQ==
</data>
</host-key>
</host-key>

<name>my-dss-key</name>
<format-identifier>ssh-dss</format-identifier>
<data>  <!-- base64 reformatted for draft -->
AAAAB3NzaC1kc3MAAACBAIg7XfGmZKjgibJEIMzj70YMVfpeewBCj89VrUS
gLSjMnP/TrXFuhzW2UIa18ePmYUxV/3g5DUD+eBSbkHMMH4ga0U5t/clqn
y73x8vG6LQqf90TaUnpRlwbrwdac7U5/BRBTtMAsmHzHhrKs7BrCepS/y8
cUBx4BCFp6aYMK/5AAAAAFQ7wetEbwghYtzBZ3xIwDdxs6m0wAAbA1BursEk
jnv5zsyHtNi4n0y9Ojysq81jPM6KopkeFa5yp2KTP9Pm/ClL05oVf1b
+HtVfYq3Q6+3sf1Qyk+gUtnisNd2AqtFQYKxtTcx4MwW61ixkYP88kkt
02t9Hs1xVxLtmogr1slnusTAbF5+Q7eq40G10DC5t7jy7DqA8IA211pZg
y5v461gt4dqhKh8ytwNyGyjBRFP6rllmslnX31MR9XwTaS7ZYP0b6HJt5M
sUfI+m7i1YaVFBlc8n1x8kkaVcxhGpNvKE2I1NSW4TlBtQphivuoE+dMMY
KaulQXqSUjixJk3LjhCQb
</data>
</host-key>
</host-key>

<name>my-call-home-x509-key</name>
<format-identifier>x509v3-rsa2048-sha256</format-identifier>
<data>  <!-- base64 reformatted for draft -->
AAAAB3NzaC1yc2EAAABABIwAAAQEAyBBL190dPUGX7Es1q7YKkw6v8wGwp+
B62zhT39C+yvsmLWigHyi0h/TGktahKpBwssawfwhvA20MF/n0yo3y3DPD
pQXnrA76H7w0j0G5206QHDYfVALKPvXrDy/6BjsR9MayOGkZ2SL6GRFS1
g71v79AIR95QxmP+1z+IDufRlpwfaGfp2AxjJLEwzAjFAlwxsXXKJ5FH/QP
mC6Gxfqgh9tJCD1gqmrXi8dXKseFUC3/ollkezqTXTV11METTuCHgWeGF
5QcX2baFdFgCnkd1SnftVoBHvnxVAseluRqgiG3fMNK4rct0D99D+GI+kZc+
vqyUdw3dPlhXP2w==
</data>
</host-key>

</fingerprint>
</host-key>
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.

```xml
<netconf-server xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-server">
  <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>
      <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>
</netconf-server>
```
<name>fw1.east.example.com</name>
<data>  <!-- base64 reformated for draft -->
AAAAAB3NzaC1yc2EAAAABIBwAAAQEA7D2lxYg3+WD97RZqZtO8bUU8QpI6g9
X1Ik2HZ28NgSiR+x2H1MC5sEjmx/B6JiouK5eBvbJE9FFV3phs162fupN6
Y4EMsosC6iqpu41dcGA3XClQ10enWG4ppdqf8tlecStmEcLw7MKFzBHK6
rNQToiqMuVuLPOwBu/54QAiUwvvyHKA8k8bkN9YxEJ1JTV1FFQmvMDADVcD
2qqPangEtwV5zInW8AEk8BbLccM/mmHucGN81axXTR3Vt965KgXF2yGB47d2
k6iOnGa3iBIOy1/5Q+08FU10+kytfgwuFgUc+Mx7aKReSIAov3owVjeBL
KWsVjD24U68qtwQ==</data>
</certificate>
</certificates>
<client-auth>
<trusted-ca-certs>
<trusted-ca-cert>
QW4gRWFzdGVyIGVnZywgZm9yIHRob3NlIHdobyBtaWdodCBsb29rICA6KQo=
</trusted-ca-cert>
</trusted-ca-certs>
<trusted-client-certs>
<trusted-client-cert>
SSBhbSB0aGUgZWdnIG1hbiwgdGhleSBhcmUgdGhlIGVnZyBtZW4uCg==
</trusted-client-cert>
<trusted-client-cert>
SSBhbSB0aGUgd2FscnVzLCBnb28gZ29vIGcnam9vYi4K
</trusted-client-cert>
</trusted-client-certs>
<cert-maps>
<cert-to-name>
<id>1</id>
<fingerprint>11:0A:05:11:00</fingerprint>
<map-type>x509c2n:SAN-Any</map-type>
</cert-to-name>
<cert-to-name>
<id>2</id>
<fingerprint>11:0A:05:11:00</fingerprint>
<map-type>x509c2n:specified</map-type>
<name>Joe Cool</name>
</cert-to-name>
</cert-maps>
</client-auth>
</tls>
</netconf-server>

Appendix B. Change Log
B.1.  00 to 01
   o Restructured document so it flows better
   o Added trusted-ca-certs and trusted-client-certs objects into the
     ietf-system-tls-auth module
B.2.  01 to 02
   o removed the "one-to-many" construct
   o removed "address" as a key field
   o removed "network-manager" terminology
   o moved open issues to github issues
   o brought TLS client auth back into model
B.3.  02 to 03
   o fixed tree diagrams and surrounding text
B.4.  03 to 04
   o reduced the number of grouping statements
   o removed psk-maps and associated feature statements
   o added ability for listen/call-home instances to specify which
     host-keys/certificates (of all listed) to use
   o clarified that last-connected should span reboots
   o added missing "objectives" for selecting which keys to use,
     authenticating client-certificates, and mapping authenticated
     client-certificates to usernames
   o clarified indirect client certificate authentication
   o added keep-alive configuration for listen connections
   o added global-level NETCONF session parameters
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