Juniper Networks | Engineering Simplicity

IN FOCUS

Junos® OS Release 20.3

Published
2020-09-29

junos

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

IN FOCUS Junos® OS Release 20.3

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement (“EULA”) posted at https://support.juniper.net/support/eula/. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

CHAPTER 1

Start Here with Junos OS Release 20.3

What You Need to Know About the In Focus Guide

Use this guide to quickly learn about the most important features in Junos OS Release 20.3 and how you can deploy them in your network.

You might also be interested in seeing the complete list of features in the Release Notes for Junos OS Release 20.3. In addition to this guide, you can find detailed information on concepts, configuration, and examples in the Junos OS documentation.

Want to tell us what you think about this guide? E-mail us at techpubs-comments@juniper.net.

Important Features in Junos OS Release 20.3

For details on these features, go to the other chapters in this guide or click the link in the feature description below.

- **Probe command to query the status of the probed interfaces (ACX Series, EX Series, MX Series, PTX Series, QFX Series, and SRX Series)**—Starting in Junos OS Release 20.3R1, you can use the **probe** command to query the status of the probed interface. The proxy interface resides on the same node as the probed interface, or it can reside on a node to which the probed interface is directly connected.

  The Probe command helps to capture the interface details such as system statistics, and interface state (active/inactive), irrespective of whether the network family address configured is IPv4 or IPv6 on the probed interfaces.

  To enable the **probe** command, configure the **extended-echo** statement under the **[edit system]** hierarchy.

  [See "How to use the Probe command" on page 10.]

- **Support for phone-home client (EX3400 Virtual Chassis and EX4300 Virtual Chassis)**—Starting in Junos OS Release 20.3R1, the phone-home client (PHC) can securely provision a Virtual Chassis without requiring user interaction. You only need to:

  - Ensure that the Virtual Chassis members have the factory-default configuration.
  - Interconnect the member switches using dedicated or default-configured Virtual Chassis ports.
  - Connect the Virtual Chassis management port or any network port to the network.
  - Power on the Virtual Chassis members.

  PHC automatically starts up on the Virtual Chassis and connects to the phone-home server (PHS). The PHS responds with bootstrapping information, including the Virtual Chassis topology, software image, and configuration. PHC upgrades each Virtual Chassis member with the new image and applies the configuration, and the Virtual Chassis is ready to go.

[See "How to Provision a Virtual Chassis Using the Phone-Home Client" on page 18.]

- **Support for TCP authentication option (TCP-AO) for BGP and LDP connections (MX Series and PTX Series)**——Starting in Junos OS Release 20.3R1, you can use TCP-AO to authenticate TCP segments exchanged during BGP and LDP sessions. It supports both IPv4 and IPv6 traffic. TCP-AO provides a framework to support multiple stronger algorithms, such as HMAC-SHA1 and AES-128, to create its message digest. TCP-AO supports up to 64 keys that can be used for a BGP or an LDP session. You can configure a new key for a BGP or LDP session during its lifetime without causing any session flap. Each key becomes active based on its configured start time.

  In earlier releases, you could use only the TCP MD5 authentication method. It supports only MD5 algorithm to create its message digest.

  [See "TCP Authentication Option (TCP-AO)" on page 24].

- **Support for LDP Tunneling over Segment Routing Traffic Engineering (MX Series, PTX Series, and ACX5448)**—Starting in Junos OS Release 20.3R1, you can tunnel LDP LSPs over Segment Routing Traffic Engineering (SR-TE) in your network. Tunneling LDP over SR-TE provides consistency and co-existence of both LDP LSPs and SR-TE LSPs.

  [See "How to Tunnel LDP LSPs over Segment Routing Traffic Engineering" on page 29.]

- **Support for IP-over-IP next-hop-based tunneling (MX Series, PTX1000, PTX10000, and QFX10000)**—Starting in Junos OS Release 20.3R1, we support an IP-over-IP encapsulation to facilitate IP overlay construction over an IP transport network. An IP network contains edge devices and core devices. To achieve higher scale and reliability among these devices, you need to use an overlay encapsulation to logically isolate the core network from the external network that the edge devices interact with. Among other supported encapsulation methods, only IP-over-IP allows transit devices to parse the inner payload and use inner packet fields for hash computation and customer edge devices to route traffic into and out of the tunnel without any throughput reduction. IP-over-IP relies on a next-hop-based infrastructure to support higher scale.

  On MX Series routers, the routing protocol daemon (rpd) sends the encapsulation header with tunnel composite next hop and the Packet Forwarding Engine finds the tunnel destination address and forwards the packet. On PTX Series routers and QFX10000 switches, rpd sends the fully resolved next-hop-based tunnel to the Packet Forwarding Engine. You can either use static configuration or a BGP protocol configuration to distribute routes and signal dynamic tunnels. You can also configure Interface based firewall filters on any transit or egress device with an action to decapsulate IP-IP packets and forward it to the main instance or to a routing-instance as required.

  See "Next-Hop-Based Dynamic Tunneling Using IP-Over-IP Encapsulation" on page 34

- **SRv6 network programming in IS-IS (MX Series with MPC7E, MPC8E and MPC9E line cards)**—Starting in Junos OS Release 20.3R1, you can configure segment routing (SR) in a core IPv6 network without an MPLS data plane. This feature is useful for service providers whose networks are predominantly IPv6 and have not deployed MPLS. Such networks depend only on the IPv6 headers and header extensions for transmitting data. This feature benefits networks that need to deploy segment routing traffic through

transit routers that do not have segment routing capability yet. In such networks, the SRv6 network programming feature can provide flexibility to leverage segment routing without deploying MPLS.

To enable SRv6 network programming in an IPv6 domain, include the **srv6** statement at the **[edit routing-options source-packet-routing]** hierarchy level.

To advertise the SRH locator with a mapped flexible algorithm, include the **algorithm** statement at the **[edit protocols isis source-packet-routing srv6 locator]** hierarchy level.

To configure topology-independent loop-free alternate backup path for SRv6 in an IS-IS network, include the **transit-srh-insert** statement at the **[edit protocols isis source-packet-routing srv6]** hierarchy level.

[See "How to Enable SRv6 Network Programming in IS-IS Networks" on page 37.]

- **Enhancements to sessions over outbound HTTPS (EX Series, MX Series, PTX Series, QFX Series, and SRX Series)**—Starting in Junos OS Release 20.3R1, devices running Junos OS with upgraded FreeBSD support the following enhancements to sessions over outbound HTTPS:
  - Connecting to multiple outbound HTTPS clients by configuring one or more clients at the **[edit system services outbound-https]** hierarchy level
  - Configuring multiple backup gRPC servers for a given outbound HTTPS client
  - Establishing a csh session
  - Establishing multiple, concurrent NETCONF and csh sessions between the device running Junos OS and an outbound HTTPS client
  - Configuring a shared secret that the outbound HTTPS client uses to authenticate the device running Junos OS
  - Authenticating the client using certificate chains in addition to self-signed certificates

  [See "How to Configure NETCONF or Shell Sessions over Outbound HTTPS" on page 44.]

CHAPTER 2

Implementation of the Probe command in Junos OS

How to use the Probe command

SUMMARY

Learn how to configure and use the Probe command.

IN THIS SECTION

Benefits of the Probe command

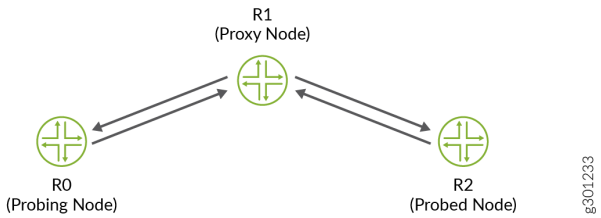
The Probe command helps to capture the interface details such as system statistics, and interface state (active/inactive), irrespective of whether the network family address configured is IPv4 or IPv6 on the probed interfaces.

What is the Probe Command?

RFC 8335, *PROBE: A Utility for Probing Interfaces*, describes the Probe utility. Probe is a network diagnostic tool similar to Ping that can be used to query the status of a probed interface on a node. The Probe command require bidirectional connectivity between the probing interface and the proxy interface. The proxy interface can reside on the same node as the probed interface, or it can reside on a node to which the probed interface is directly connected. Probe uses ICMP Extended Echo/Reply messages for communication between the probing interface and the proxy interface. The Probe command is used to probe the details of the interfaces on the probed node that is directly connected to the proxy, where the proxy learns the details from the ARP or NDP entries of those interfaces. This utility helps in scenarios where bidirectional connectivity between the probing and probed interfaces is lacking. For example, if the probed interface is an unnumbered interface or if the probed interface is in a different network family.

The following figure shows R0 as the probing node, R1 as the proxy node, and R2 as the probed node:

Figure 1: Probing, Proxy, and Probed Nodes



Enabling the Probe command

To enable the **probe** command, configure the **extended-echo** configuration statement at the **[edit system]** hierarchy level on the proxy node and the probed node.

```
[edit]
user@host# set system extended-echo
```

> **NOTE:** You do not need to enable the **extended-echo** configuration statement on the probing node.

Using the Probe command

You can probe using three different options:

- By using IP address.
- By using interface names.
- By using the interface index.

The following examples shows how to use the **probe** command. Consider R0 as the probing node, R1 as the proxy node, and R2 as the probed node and change the details of the interfaces and IP addresses to match your network configuration.

Purpose

Use case 1: Probing when proxy node (R1) and the probed node (R2) are two different nodes.

Query for the status of the probed IP address through the proxy IP address.

Action

From operational mode, probe for the status of the IP address 198.52.102.36 of the probed node (R2) using the **by-address** option with the proxy IP address 198.52.100.1 of the proxy node (R1). Count 1 indicates that the number of probe requests sent is 1.

user@R0>**probe 198.52.100.1 by-address 198.52.102.36 count 1**

```
PROBE 198.52.100.1 (198.52.100.1):
35 bytes from 198.52.100.1: icmp_seq=0 ttl=255 code=2 state=0 active=0 IPv4=0
IPv6=0 time=5.195 ms

--- 198.52.100.1 probe statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 5.195/5.195/5.195/0.000 ms
```

Meaning

The proxy IP address, 198.52.100.1 displays the status of the probed IP address, 198.52.102.36. The output is verified with the following probe packet statistics:

- **code=2** indicates that IP address is not configured on the probed interface.
- **state=0** indicates that at the time of probing, the state of the IPv6 or IPv4 node is reachable.
- **active=0** indicates that the probed node is inactive as the IPv4 or IPv6 address is not available or configured on the probed node. That is, the probed IP address, 198.52.102.36 is not configured on the probed node (R2).
- **IPv4=0** and **IPv6=0** indicates that the IPv4 and IPv6 addresses are not configured.
- **time=*milliseconds* ms** indicates the time taken to transmit the packets.
- **ttl** is the time to live value (maximum IPv6 hop-limit-value).

Purpose

Use Case 2: Probing when proxy node and the probed node are the same.

Query for the status of the probed IP address through the proxy IP address (here, the probed IP address and the proxy IP address are configured on the same node, R1)

Action

From operational mode, probe for the status of the probed IP address 198.52.102.2 using the **by-address** option with the proxy IP address 198.52.100.1 at the proxy node (R1). Count 1 indicates that the number of probe requests sent is 1.

user@R0>**probe 198.52.100.1 by-address 198.52.102.2 count 1**

```
PROBE 198.52.100.1 (198.52.100.1):
32 bytes from 198.52.100.1: icmp_seq=0 ttl=255 code=0 state=0 active=1 IPv4=1
IPv6=1 time=4.908 ms

--- 198.52.100.1 probe statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 5.195/5.195/5.195/0.000 ms
```

Meaning

The proxy IP address, 198.52.100.1 displays the status of the probed IP address, 198.52.102.2. The output is verified with the following probe packet statistics:

- **code=0** indicates that the sent packet is received by the probed node. The code value will be 0 as proxy and probed nodes are the same.
- **state=0** indicates that at the time of probing the state of the IPv6 or IPv4 node is reachable.
- **active=1** indicates that the probed node is active and either the IPv4 or IPv6 addresses or both are configured and available on the probed node.
- **IPv4=1** and **IPv6=1** indicates that the IPv4 and IPv6 addresses are configured and available on the probed node.
- **time=*milliseconds* ms** indicates the time taken to transmit the packets.
- **ttl** is the time to live value (maximum IPv6 hop-limit-value).

Purpose

Query for the status of the IPv4 or IPv6 address using the interface index of the probed interface.

Action

From operational mode, probe for the status of the IPv4 or IPv6 address using the **by-index** option to specify the interface index of the probed interface.

user@R0>**probe 198.52.100.1 by-index 1037**

```
PROBE 198.52.100.1 (198.52.100.1):
28 bytes from 198.52.100.1: icmp_seq=0 ttl=255 code=0 state=0 active=1 IPv4=1
IPv6=0 time=0.871 ms
28 bytes from 198.52.100.1: icmp_seq=1 ttl=255 code=0 state=0 active=1 IPv4=1
IPv6=0 time=0.864 ms


--- 198.52.100.1 probe statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.864/0.867/0.871/0.004 ms
```

Meaning

The proxy IP address, 198.52.100.1 displays the status of the probed interface using the interface index-id, 1037. The output is verified with the following probe packet statistics:

- **code=0** indicates that the sent packet is received by the probed node.
- **state=0** indicates that at the time of probing the state of the IPv6 or IPv4 node is reachable.
- **active=1** indicates that the probed node is active and either the IPv4 or the IPv6 address is available.
- **IPv4=1** indicates that the IPv4 address is configured and available. **IPv6=0** indicates that the IPv6 address is not configured.
- **time=*milliseconds* ms** indicates the time taken to transmit the packets.
- **ttl** is the time to live value (maximum IPv6 hop-limit-value).

Purpose

Query the proxy node (R1) for information about the remote node using the remote address.

Action

From operational mode, probe for the status of the probed node and verify using the **by-remote-address** option with the IP address of a remote device. Count 1 indicates that the number of probe requests sent is 1.

user@R0>**probe 198.52.100.1 by-remote-address 2001:65::2 count 1**

```
PROBE 198.52.100.1 (198.52.100.1):
44 bytes from 198.52.100.1: icmp_seq=0 ttl=255 code=0 state=2 active=0 IPv4=0
IPv6=0 time=6.691 ms

--- 198.52.100.1 probe statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 6.691/6.691/6.691/0.000 ms
```

Meaning

The proxy IP address, 198.52.100.1 displays the status of the probed node using the IP address, 2001:65::2 of the remote device. The **by-remote-address** option is used to determine the status of the remote node only. The output is verified with the following probe packet statistics:

- **code=0** indicates that the sent packet is received by the probed node.
- **state=2** indicates that the connection is established with a remote node where the IPv4 and the IPv6 address is not available.
- **active=0** indicates that the probed node is inactive.
- **IPv4=0** and **IPv6=0** indicates that the IPv4 and the IPv6 addresses are not configured or available. Note that when you probe using the **by-remote-address** option, the IPv4 or the IPv6 address is not available.
- **time=*milliseconds* ms** indicates the time taken to transmit the packets.
- **ttl** is the time to live value (maximum IPv6 hop-limit-value).

Purpose

Query for the status of the probed node using the interface name.

Action

From operational mode, probe for the status of the probed interface using the interface name ge-1/2/1.1 and verify the output through the IP address of the proxy node 198.52.100.1. Count 1 indicates that the number of probe requests sent is 1.

user@R0>**probe 198.52.100.1 by-name ge-1/2/1.1 count 1**

```
PROBE 198.52.100.1 (198.52.100.1):
34 bytes from 198.52.100.1: icmp_seq=0 ttl=255 code=0 state=0 active=1 IPv4=1
IPv6=0 time=0.807 ms

--- 198.52.100.1 probe statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.807/0.807/0.807/0.000 ms
```

Meaning

The proxy IP address, 198.52.100.1 displays the status of the device with reference to the interface ge-1/2/1.1 of the device on the probed node. The output is verified with the following probe packet statistics:

- **code=0** indicates that the connection is successfully established and the ICMP reply packet has no error.
- **state=0** indicates that the proxy is able to establish a connection with the probed node.
- **active=1** indicates that the probed node is active and either the IPv4 or the IPv6 address configured is available.
- **IPv4=1** indicates that the IPv4 address is configured and available. **IPv6=0** indicates that the IPv6 address is not configured.
- **time=*milliseconds* ms** indicates the time taken to transmit the packets.
- **ttl** is the time to live value (maximum IPv6 hop-limit-value).

SEE ALSO

> *extended-echo*
>
> *probe*

3

CHAPTER

Phone-Home Client on a Virtual Chassis

How to Provision a Virtual Chassis Using the Phone-Home Client

SUMMARY

Phone-home provisioning on a Virtual Chassis is a form of zero-touch provisioning (ZTP). The phone-home client (PHC) on the Virtual Chassis gets bootstrap information over the network from a phone-home server (PHS) and provisions the Virtual Chassis. The only user intervention required on the client side is to physically wire the Virtual Chassis members together and connect any port on the Virtual Chassis to the network.

IN THIS SECTION

Overview of Phone-Home Provisioning for a Virtual Chassis

IN THIS SECTION

With phone-home provisioning, a phone-home client (PHC) on a device initially provisions the device with a software image and configuration from a central network management data source called the phone-home server (PHS), requiring little or no user intervention at the remote site.

A Virtual Chassis consists of a set of devices interconnected together using ports called Virtual Chassis ports (VCPs). You configure and manage the Virtual Chassis as a single device. Starting with Junos OS Release 20.3R1, we've made extensions to the phone-home provisioning process for a standalone device so it can also work on a Virtual Chassis. The PHC on a Virtual Chassis requires extra steps to coordinate and manage bootstrapping the member devices.

The PHS is usually part of a network management system (NMS) that supports phone-home provisioning. Your network administrator enters the intended provisioning data that directs how devices and Virtual Chassis at remote sites should be set up. Your organization might have more than one PHS for redundancy.

You can check Feature Explorer and search for **phone-home** to see the Virtual Chassis platforms that support phone-home provisioning.

Benefits of Phone-Home Provisioning on a Virtual Chassis

- Simplifies provisioning by launching the process automatically from the remote site, while securely obtaining bootstrap information from a central management system (the PHS) on your network or in the cloud.
- Doesn't require in-depth experience with the Junos OS CLI to coordinate the provisioning of multiple devices that make up a Virtual Chassis.

Overview of the Phone-Home Provisioning Process on a Virtual Chassis

On a Virtual Chassis that supports phone-home provisioning, for the process to work, you must set up the Virtual Chassis according to the requirements outlined in "How To Enable Phone-Home Provisioning on a Virtual Chassis" on page 20.

When the Virtual Chassis initially forms, the PHC process starts up automatically on the Virtual Chassis master member and takes it from there:

1. The PHC connects to a PHS.

   The PHC sends a provisioning request to a default redirect server URL, https://redirect.juniper.net, which redirects the request to an available PHS controlled by your network administrator or NMS. This step is the same as phone-home provisioning on a single device.

2. The PHS responds to the PHC provisioning request with the bootstrapping information, which includes the intended Virtual Chassis topology, software image, and configuration.

3. The PHC provisions the Virtual Chassis as specified by the PHS.

   Provisioning includes steps such as:

   - Validate the Virtual Chassis topology.
   - Upgrade the software image sequentially on all of the member devices if needed.
   - Run any pre-configuration or post-configuration staging scripts.
   - Commit a new configuration on the Virtual Chassis.

The PHC sends status notifications to the PHS during the bootstrapping process, so the network administrator can verify the process completes successfully.

The PHC also logs status locally in the system log files on the Virtual Chassis. If needed, you can view log files in the Junos OS CLI, and use Junos OS CLI commands to see Virtual Chassis and VCP connection status.

SEE ALSO

> *Obtaining Configurations and Software Image Without User Intervention Using Phone-Home Client*
>
> *Zero Touch Provisioning*

How To Enable Phone-Home Provisioning on a Virtual Chassis

On a Virtual Chassis that supports phone-home provisioning, if you set up the Virtual Chassis according to the steps listed here, a phone-home client (PHC) process starts up automatically on the Virtual Chassis master member.

To enable phone-home provisioning on a Virtual Chassis:

1. Ensure that all Virtual Chassis members have the factory-default configuration and are powered off.

   You can run the **request system zeroize** Junos OS CLI command to return a device to its factory-default state.

2. Interconnect the Virtual Chassis members in a ring topology using only dedicated or default-configured Virtual Chassis ports (VCPs) on each member device.

   Keep in mind that the PHC process works only if the Virtual Chassis is initially formed with VCPs that do not need to be explicitly configured (dedicated VCPs or ports that are VCPs in the factory-default configuration). See *VCP Options by Switch Type* for details on which ports are dedicated and default-configured VCPs on different devices that support Virtual Chassis. See the hardware guide for the device to locate those ports on the device.

3. Connect the Virtual Chassis management interface (**me0**) or any network-facing port on any Virtual Chassis member to the network.

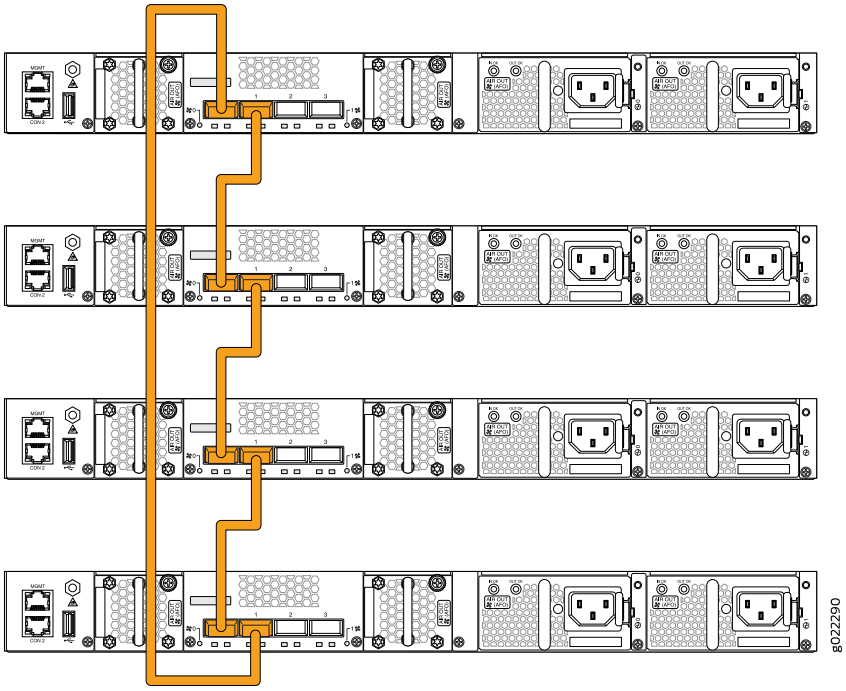
   After the PHC starts up on the Virtual Chassis, it uses this connection to access a PHS over the network and retrieve the bootstrapping information for this Virtual Chassis.

For details about how the management interfaces work on a Virtual Chassis, see *Understanding Global Management of a Virtual Chassis*.

4. Power on the members of the Virtual Chassis.

Figure 2 on page 21 shows an example of a Virtual Chassis topology that can support phone-home provisioning—a four-member EX4300 Virtual Chassis cabled in a ring topology using default-configured VCPs (in this case, two of the 40-Gigabit Ethernet QSFP+ ports on each device).

Figure 2: Sample Virtual Chassis That Can Support Phone-Home Provisioning



Usually you don't need to do anything else for the phone-home provisioning process to proceed and complete successfully. If you don't see successful completion status or the Virtual Chassis isn't up and operating as expected at the end of the process, read on to learn details about how the PHC works to help troubleshoot the issues.

SEE ALSO

- *Understanding Virtual Chassis Components*
- *Virtual Chassis Overview for Switches*

WHAT'S NEXT

> If provisioning completes successfully and your Virtual Chassis is up and running with the expected software version and configuration from your network management system, you're done!
>
> If you notice any unexpected results or want to know more about how the PHC works, see *Provision a Virtual Chassis Using the Phone-Home Client*. The details there can help you troubleshoot problems and track what's going on during the process.

4

CHAPTER

TCP Authentication Option (TCP-AO)

TCP Authentication Option (TCP-AO)

SUMMARY

Learn about TCP Authentication Option (TCP-AO) for BGP and LDP Sessions.

IN THIS SECTION

TCP Authentication Option (TCP-AO) for BGP and LDP Sessions

IN THIS SECTION

Benefits of TCP-AO

TCP-AO provides the following benefits over TCP MD5:

- **Stronger algorithms**—Supports multiple stronger authentication algorithms such as HMAC-SHA-1-96 and AES-128-CMAC-96 (mandated by *RFC5925, The TCP Authentication Option*). HMAC-SHA-1-96 is a hash-based MAC and AES-128-CMAC-96 is a cipher-based MAC, thus making the message digest more complex and secure than the digest created by using MD5 algorithm.
- **Two-folded security**—In TCP-AO method, the configured Authentication algorithm used in two stages: once to generate an internal traffic key from a user-configured key and then to generate a message digest using the generated traffic key, whereas in TCP MD5 method, the MD5 algorithm generates a message digest using its user-configured key.
- **Better Key Management and Agility**—You can configure up to 64 keys for a session and you can add them at any time during the lifetime of a session. It provides a simple key coordination mechanism by giving the ability to change keys (move from one key to another) within the same connection without

causing any TCP connection closure. Changing TCP MD5 keys during an established connection might cause a flap or restart in the connection.

- **Suitable for long-lived connections**—More suitable for long-lived connections for routing protocols such as BGP and LDP and across repeated instances of a single connection.

What is TCP-AO?

BGP and LDP protocols use TCP for transport. TCP-AO is a new authentication method proposed through *RFC5925*, *The TCP Authentication Option* to enhance the security and authenticity of TCP segments exchanged during BGP and LDP sessions. It also supports both IPv4 and IPv6 traffic.

TCP-AO provides a framework to:

- Add multiple stronger algorithms under it, such as HMAC-SHA1 and AES-128 to create internal traffic key and message digest.
- Add a new user-configured key to be used to re-generate internal traffic keys for an established connection and a mechanism to synchronize key change between BGP or LDP peers.

In releases prior to Junos OS release 20.3R1, TCP MD5 is the only authentication method. It supports a single MAC algorithm to create its MAC digest. TCP MD5 is defined in *RFC2385*, *Protection of BGP Sessions via the TCP MD5 Signature Option*.

NOTE:

- Although TCP-AO and TCP MD5 authentication methods are both supported now, you cannot use both at the same time for a given connection.
- TCP-AO supports Nonstop Active Routing.

- To configure a key chain for TCP-AO (with one key), set the following statement at the **[edit security]** hierarchy level.

  user@router# set authentication-key-chains key-chain *key-chain* key *id* secret *secretpassword* start-time *YYYY-MM-DD.HH:MM:SS* algorithm *ao* ao-attribute send-id *send-id* recv-id *recv-id* cryptographic-algorithm *cryptographic-algorithm* tcp-ao-option *enabled/disabled*

- To apply TCP-AO to a BGP session (with the configured key chain), set the following statement at the **[edit protocols]** hierarchy level.

  user@router# set bgp group *group* neighbor *neighbor* authentication-algorithm *ao*

  user@router# set bgp group *group* neighbor *neighbor* authentication-key-chain *key-chain*

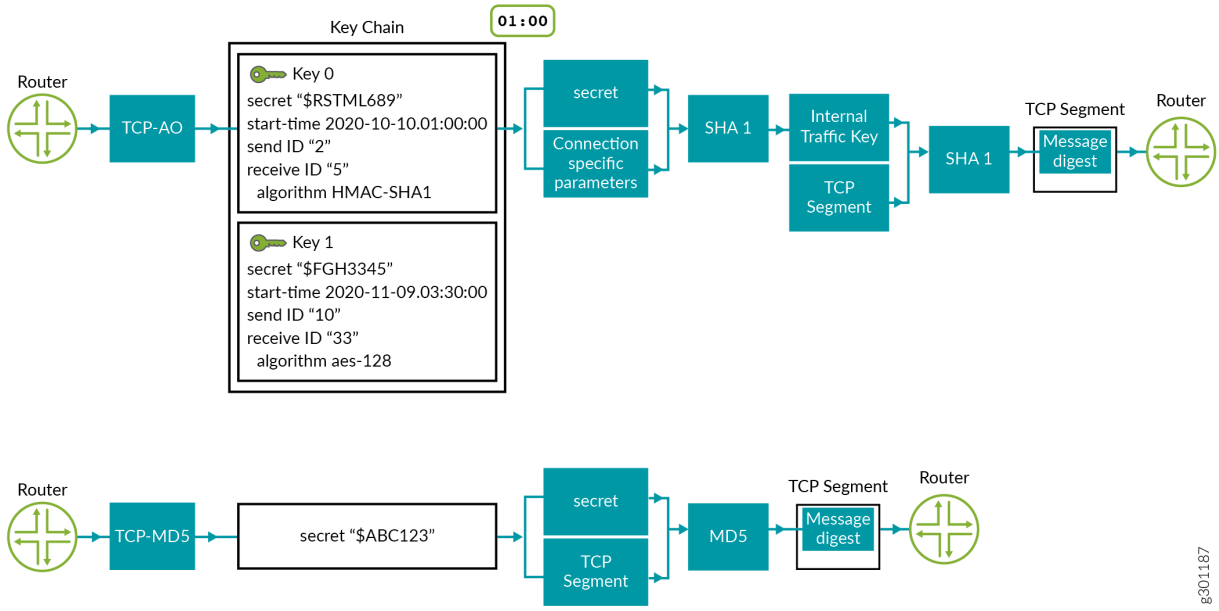
- To apply TCP-AO to an LDP session (with the configured key chain), set the following statement at the **[edit protocols]** hierarchy level.

  user@router# set ldp session *session* authentication-algorithm *ao*

user@router# set ldp session *session* authentication-key-chain *key-chain*

The following diagram explains the difference between TCP Authentication Option (TCP-AO) and TCP MD5 authentication. The first flow shows the configuration for TCP-AO and second flow shows the configuration flow for TCP-MD5.

Figure 3: TCP-AO in comparison with TCP MD5



Below is an explanation of the configuration flows shown in Figure 1:

- **TCP-AO**—User has configured two keys in its key chain- key 0 and key 1 with its required parameters. It supports two algorithms: HMAC SHA1 and AES-128 (mandated per RFC5925). TCP fetches key 0, the key that is active at that time. In the diagram, key 0 is configured with HMAC-SHA1.

  SHA1 takes the "secret" (from the key 0 configuration) and connection specific parameters for encryption and generates an Internal Traffic Key.

  SHA1 again encrypts the Internal Traffic Key and the TCP segment to generate the message digest, which is then copied to the TCP-AO MAC digest field of TCP-AO option in TCP segment and is sent to the receiving device.

- **TCP-MD5**—User has configured one key here as TCP MD5 supports only one key for a connection. It only supports MD5 algorithm. MD5 algorithm takes the "secret" from the key and the TCP segment for encryption and generates a message digest. This message digest is then copied to MD5 digest field in the TCP segment and is sent to the receiving device.

WHAT'S NEXT

For more information, see the Configure TCP Authentication Option (TCP-AO)

5

CHAPTER

Tunnel LDP LSPs over Segment Routing Traffic Engineering

How to Tunnel LDP LSPs over Segment Routing Traffic Engineering

SUMMARY

Learn how to tunnel LDP LSPs over Segment Routing Traffic Engineering (SR-TE) in your network.

IN THIS SECTION

Tunneling LDP over SR-TE

IN THIS SECTION

Learn about the benefits and overview of tunneling LDP over SR-TE.

Benefits of Tunneling LDP over SR-TE

- Enables seamless migration of LDP over SR-TE in the core network.
- Provides flexible connectivity options to accommodate multiple topologies, protocols, and domains.
- Enables interoperability between LDP and SR capable devices.
- Leverages SR-TE load sharing capabilities.
- Provides faster restoration of network connectivity using TI-LFA within the SR-TE domain. SR using TI-LFA routes the traffic instantly to a backup or an alternate path if the primary path fails or becomes unavailable.

Tunneling LDP over SR-TE Overview

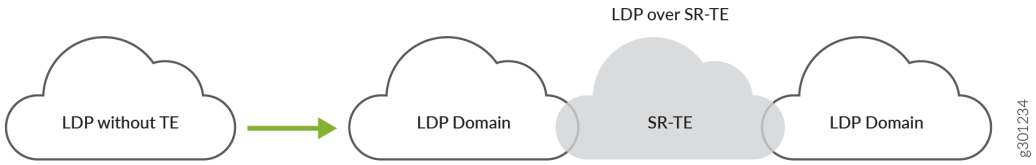
In a typical service provider network, it is common for service providers to use MPLS signaling transport protocols, such as LDP, at the edge and core routers. As a service provider, you might be looking at gradually migrating or deploying segment routing traffic engineering (SR-TE), also called as SPRING, in the core network, eliminating the need for MPLS signaling protocols, such as LDP.

You might have some routers in your network that do not support SR capabilities and you would want to continue using those routers (running LDP) without a need for an upgrade. In such scenarios, the LDP over SR-TE tunneling feature provides the interoperability benefit of using the routers that are not SR capable (running LDP) with routers that are SR capable (running SR-TE).

The LDP LSPs are tunneled through the SR-TE network, enabling interworking of LDP LSPs with SR-TE LSPs. For example, if you have LDP domains on the provider edge network and SR-TE in the core network, then you can connect the LDP domains over SR-TE, as shown in Figure 4 on page 30. You can continue to have IGP (OSPF or IS-IS) in the traffic engineered core and in the surrounding LDP domains.

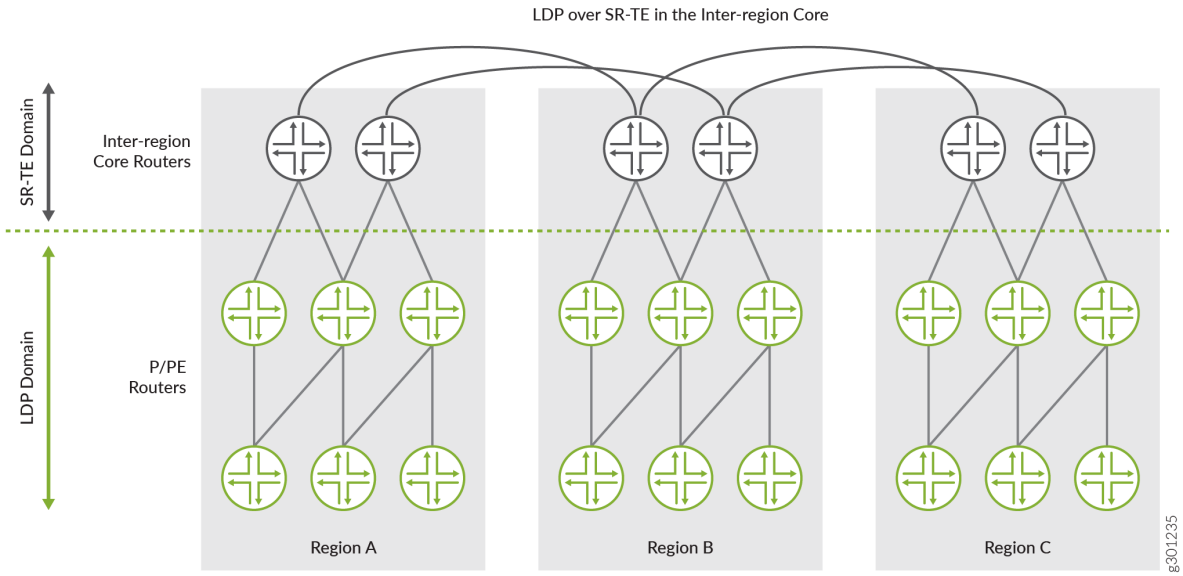
Tunneling LDP over SR-TE provides consistency and co-existence of both LDP LSPs and SR-TE LSPs.

Figure 4: Interconnect LDP Domains over SR-TE in the Core Network



You can also tunnel LDP over SR-TE between LDP domains connected to inter-region core networks. For example, if you have multiple regional LDP domains connected to the regional SR-TE core networks, you can tunnel LDP across the inter-region SR-TE core networks, as shown in Figure 5 on page 31.

Figure 5: LDP over SR-TE between Inter-region Core Networks



In Figure 5 on page 31, you have three regional networks (A, B, and C) running LDP. These regional LDP domains are connected to their respective regional core networks running SR-TE. The regional SR-TE core networks are further interconnected to other regional SR-TE core networks (inter-region core network). You can tunnel LDP over these inter-region SR-TE core networks and deploy services, such as Layer 3 VPN, seamlessly. This scenario could be used in a mobile backhaul network, where core aggregation can run LDP over SR-TE and access can run LDP only.

To enable LDP tunneling over SR-TE, you need to configure the following statements:

- **ldp-tunneling** at the [**edit protocols source-packet-routing source-routing-path *source-routing-path-name***] hierarchy level enables LDP tunneling over SR-TE.
- **spring-te** at the [**edit protocols isis traffic-engineering tunnel-source-protocol**] hierarchy level selects LDP over SR-TE LSPs as the tunnel source protocol.

You can configure more than one tunnel source protocol for IGPs to create shortcut routes. When more than one tunnel source protocol is configured and if the tunnels from more than one protocol is available to a destination, the tunnel with the best preferred route is established. For example, if the core network has both RSVP LSPs and SR-TE LSPs and LDP tunneling is enabled for both RSVP and SR-TE LSPs, then the **tunnel-source-protocol** configuration selects the tunnel based on the preference value. The tunnel with the lowest preference value is most preferred. You can override this route preference with a specific protocol for all destinations by configuring the preference value, as shown in the following example:

```
[edit]
user@host#set protocols isis traffic-engineering tunnel-source-protocol spring-te preference 2
user@host#set protocols isis traffic-engineering tunnel-source-protocol rsvp preference 5
```

In this example, you can see the preference value configured for the SR-TE tunnel source protocol is 2 and the preference value for RSVP tunnel source protocol is 5. In this case, SR-TE tunnel is the most preferred because it has the lowest preference value as compared to RSVP tunnel source protocol.

> **NOTE:** It is not mandatory to configure the tunnel source protocol preference value. If more than one tunnel source protocol has the same preference value, then the tunnel is established based on the preferred route to the destination.

The LDP target session is established and is triggered when the LSP comes up. The LSP session stays until the LDP tunneling (**ldp-tunneling**) configuration is removed, or the LSP is removed from the configuration.

> **NOTE:** Junos OS currently does not support LDP over colored SR-TE LSPs.

WHAT'S NEXT

- For an example on tunneling LDP over SR-TE, see Example: Tunneling LDP over SR-TE.
- For information on Label Distribution Protocol (LDP), see LDP Overview.
- For information on Tunneling LDP LSPs in RSVP LSPs, see Tunneling LDP LSPs in RSVP LSPs.
- For information on configuring segment routing label block (SRGB) label range for segment packet routing in networking (SPRING) or segment routing (SR) for the IS-IS protocol, see Example: Configuring Segment Routing Global Blocks in SPRING for IS-IS to Increase Network Speed.

CHAPTER 6

Next-Hop-Based Dynamic Tunneling Using IP-Over-IP Encapsulation

Next-Hop-Based Dynamic Tunneling Using IP-Over-IP Encapsulation

SUMMARY

Learn about the benefits and overview of next-hop-based dynamic tunneling with IP-over-IP encapsulation.

IN THIS SECTION

Next-Hop-Based Dynamic Tunneling Using IP-Over-IP Encapsulation

SUMMARY

Benefits

IP-over-IP tunneling provides the following benefits:

- **Alternative to MPLS over UDP**—Can be used as an alternative to MPLS-over-UDP tunneling to provide IP service wherein there is a dedicated device per service.
- **Ability to steer specific traffic**—Enables smooth migration when MPLS and IP networks co-exist because routes can be filtered to steer specific traffic over IP tunnels as opposed to MPLS tunnels.
- **Ability to support tunnels at increasing scale**—Dynamic tunnel creation using BGP control plane can facilitate tunnel creation at scale.

What is IP-over-IP Dynamic Next Hop-based Tunneling?

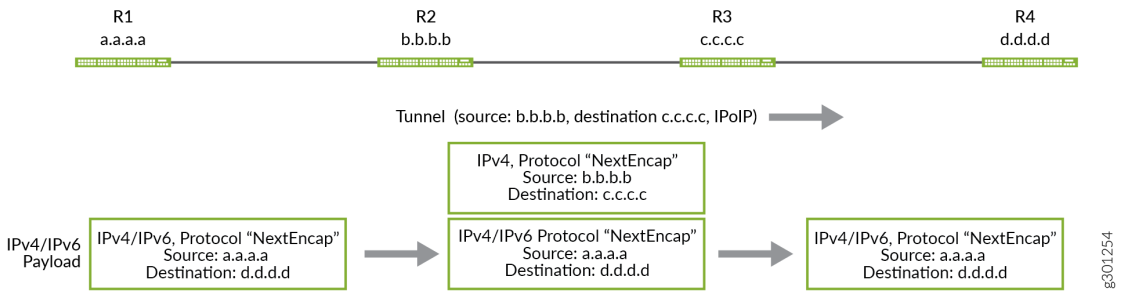
An IP network contains edge devices and core devices. To achieve higher scale and reliability among these devices, you need to logically isolate the core network from the external network that the edge devices interact with, by using an overlay encapsulation.

Starting in Junos OS Release 20.3R1, we support an IP-over-IP encapsulation to facilitate IP overlay construction over IP transport network. IP over IP relies on a next hop-based infrastructure to support a higher scale. The feature supports IPv4 encapsulation of IPv6 and IPv4 payload. Among the other overlay encapsulations supported, IP-over-IP encapsulation is the only kind that allows:

- transit devices to parse the inner payload and use inner packet fields for hash computation
- customer edge devices to route traffic into and out of the tunnel without any throughput reduction

On MX Series routers, routing protocol daemon (RPD) sends the encapsulation header with tunnel composite nexthop and the Packet Forwarding Engine (PFE) finds the tunnel destination address and forwards the packet. On PTX Series routers and QFX10000 switches, RPD sends fully resolved next hop-based tunnel to the Packet Forwarding Engine. BGP protocol is used to distribute routes and signal dynamic tunnels.

The following illustration depicts how IPv4 or IPv6 traffic are sent from R-1 to R-5 through an IP over IP tunnel established between R-2 and R-4:



SEE ALSO

WHAT'S NEXT

See Example: Configuring Next-Hop-Based IP-Over-IP Dynamic Tunnels

CHAPTER 7

SRv6 Network Programming

How to Enable SRv6 Network Programming in IS-IS Networks

SUMMARY

Learn about enabling SRv6 network programming for the IS-IS protocol.

IN THIS SECTION

Understanding SRv6 Network Programming in IS-IS Networks

IN THIS SECTION

Benefits of SRv6 Network Programming

SRv6 Network Programming provides the following benefits in an IPv6 network:

- Network Programming depends entirely on the IPv6 header and the header extension to transport a packet, eliminating protocols such as MPLS. This ensures a seamless deployment without any major hardware or software upgrade in a core IPv6 network.
- IPv4 packets can be transported through an SRv6 ingress node even if the transit routers are not SRv6-capable, thereby eliminating the need to deploy segment routing on all nodes in an IPv6 network.

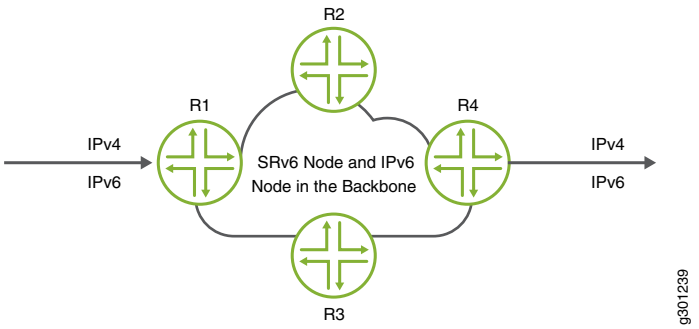
- Junos OS supports all function behaviors on a single SID and can inter-operate in the insert mode and the encapsulation mode. This allows a single device to simultaneously play the provider (P) router and the provider edge (PE) router roles.

SRv6 Networking Programming Overview

Network Programming is the capability of a network to encode a network program into individual instructions that are then inserted into the IPv6 packet headers. The IPv6 packet carrying the network instructions explicitly tells the network about the precise SRv6 nodes available for packet processing. The network instruction is the SRv6 segment identifier (SID) that is represented by a 128-bit IPv6 address. The IS-IS protocol encodes the network instructions in IPv6 packet headers and distributes them through the network. Along with the addressing, network instructions define a particular task or function for each SRv6-capable node in the SRv6 network.

> **NOTE:** Starting in Junos OS Release 20.3R1, you can configure segment routing in a core IPv6 network without an MPLS data plane on MX Series devices with MPC7E, MPC8E and MPC9E line cards.

This feature is useful for service providers whose networks are predominantly IPv6 and have not deployed MPLS. Such networks depend only on IPv6 headers and header extensions for transmitting data. This feature also benefits networks that need to deploy segment routing traffic through transit routers that do not have segment routing capability yet. In such networks, the SRv6 network programming feature can provide flexibility to leverage segment routing without deploying MPLS.



What is a Segment Routing Extension Header (SRH)?

A Segment Identifier represents a specific segment in a segment routing domain. In an IPv6 network, the SID-type used is a 128-bit IPv6 address also referenced as SRv6 Segment or SRv6 SID. SRv6 stacks up these IPv6 addresses instead of MPLS labels in a segment routing extension header. Segment Routing

Extension Header (SRH) is a type of IPv6 routing extension header. Typically, the SRH contains a segment list encoded as an SRv6 SID. An SRv6 SID consists of the following parts:

- **Locator**— Locator is the first part of a SID that consists of the most significant bits representing the address of a particular SRv6 node. The locator is very similar to a network address that provides a route to its parent node. The IS-IS protocol installs the locator route in the **inet6.0** routing table. IS-IS routes the segment to its parent node, which subsequently performs a function defined in the other part of the SRv6 SID. You can also specify the algorithm associated with this locator. You can define a flexible algorithm as per your network requirements.
- **Function**—The other part of the SID defines a function that is performed locally on the node that is specified by the locator. There are several functions that have already been defined in the Internet draft draft-ietf-spring-srv6-network-programming-07draft, *SRv6 Network Programming* . However, we have implemented the following functions that are signalled in IS-IS. IS-IS installs these function SIDs in the **inet6.0** routing table.
  - **End**— An endpoint function for SRv6 instantiation of a Prefix SID. It does not allow for decapsulation of an outer header for the removal of an SRH. Therefore, an End SID cannot be the last SID of a SID list and cannot be the Destination Address (DA) of a packet without an SRH (unless combined with the PSP, USP or USD flavors)
  - **End.X**— An endpoint X function is SRv6 instantiation of an adjacent SID. It is a variant of the endpoint function with Layer 3 cross-connect to an array of Layer 3 adjacencies.

  You can specify End SID behavior such as Penultimate Segment Pop (PSP), Ultimate Segment Pop (USP) or Ultimate Segment Decapsulation (USD).
  - **PSP**— When the last SID is written in the destination address, the End and End.X functions with the PSP flavor pop the top-most SRH. Subsequent stacked SRHs may be present but are not processed as part of the function.
  - **USP**— When the next header is SRH and there are no more segments left, the IS-IS protocol pops the top SRH, looks up the updated destination address and forwards the packet based on match table entry.
  - **USD**— When the next Header in the packet is 41 or is SRH and there are no more segments left then IS-IS pops the outer IPv6 header and its extension headers, looks up the exposed inner IP destination address and forwards the packet to the matched table entry.

> **NOTE:** The size of the locator and function is flexible and you can customize their size per your requirements. You must configure the locator before you define the functions. Each locator can advertise multiple end SIDs and end.X SIDs associated with it. Ensure that the locator and SIDs belong to the same subnet to avoid commit error.

For example, you can have an SRv6 SID where 2019:AC05:FF01:FF01: is the locator and A000:B000:C000:A000 is the function:

Table 1: 128-bit SRv6 SID

Locator	Function
2019:AC05:FF01:FF01	A000:B000:C000:A000

Flexible Algorithm for SRv6 Dataplane

In a core IPv6 domain configured with segment routing you can define flexible algorithms that compute paths using different parameters and link constraints based on your requirements. For example, you can define a flexible algorithm that computes a path to minimize IGP metric and define another flexible algorithm to compute a path based on traffic engineering metric to divide the network into separate planes. You can configure the flexible algorithm locators to steer packets along the constraint-based paths in an SRv6 domain.

To configure flexible algorithm for SRv6, see *How to Configure Flexible Algorithm in IS-IS for Segment Routing Traffic Engineering*

To advertise the flexible algorithm mapped to the locator, include the **algorithm** option at the **[edit protocols isis segment-packet-routing srv6 locator]** hierarchy level. The mapped flexible algorithm is applied to End SIDs and End-X-SID under SRv6 locators.

> **NOTE:** If a node is participating in a specific flexible algorithm it would apply to both SR MPLS and SRv6 nodes. You cannot define flexible algorithms specifically for either SR MPLS or SRv6.

For ingress traffic, Junos OS uses the encapsulation mode by default. Therefore the destination needs to have USD capable SIDs. Other SRH anchor nodes in the flexible algorithm path can be of any flavor.

For transit traffic in the insert mode, the last anchor node for the flexible algorithm path must have a PSP-capable SID. In the absence of the PSP-capable SID, IS-IS does not download a path through that anchor node. In such cases, IS-IS downloads other ECMP paths with the appropriate flavored SIDs.

TI-LFA for SRv6

Topology Independent- Loop Free Alternate (TI-LFA) establishes a Fast Reroute (FRR) path that is aligned to a post-convergence path. An SRv6-capable node inserts a single segment into the IPv6 header or multiple segments into the SRH. Multiple SRHs can significantly raise the encapsulation overhead that can sometimes be more than the actual packet payload. Therefore, by default, Junos OS supports SRv6 tunnel encapsulation

with reduced SRH. The point-of-local repair (PLR) adds the FRR path information to the SRH containing the SRv6 SIDs.

The TI-LFA backup path is represented as a group of SRv6 SIDs inside an SRH. At the ingress router, IS-IS encapsulates the SRH in a fresh IPv6 header. However, at transit routers, IS-IS inserts the SRH into the data traffic in the following manner:

- **Insert Mode**— IS-IS inserts an SRH as the next header in the original IPv6 packet header and modifies the next header according to the value of the SRH. The IPv6 destination address is replaced with the IPv6 address of the first SID in the segment list and the original IPv6 destination address is carried in the SRH header as the last segment in the list. To enable the insert mode at transit routers, include the **transit-srh-insert** statement at the **[edit protocols isis source-packet-routing srv6]** hierarchy level.
- **Encap Mode**— In the encap mode, the original IPv6 packet is encapsulated and transported as the inner packet of an IPv6-in-IPv6 encapsulated packet. The outer IPv6 packet carries the SRH with the segment list. The original IPv6 packet travels unmodified in the network. By default, Junos OS supports SRv6 tunnel encapsulation in reduced SRH. However, you can choose one of the following tunnel encapsulation methods:
  - **Reduced SRH**— With the reduced SRH mode, because there is only one SID, there is no SRH added and the last SID is copied into the IPV6 destination address. You cannot preserve the entire SID list in the SRH with a reduced SRH.
  - **Non-reduced SRH**— You can configure the tunnel encapsulation mode and might still want to preserve the entire SID list in the SRH.

    To configure non-reduced SRH, include the **no-reduced-srh** statement at the **[edit routing-options source-packet-routing srv6 locator]** hierarchy level.

> **NOTE:** Fate-sharing configuration is currently not supported in IPv6 only networks. Also, SRv6 TI-LFA does not take Shared Risk Link Group (SRLG) into consideration when computing backup paths. For more information on TI-LFA, see *Understanding Topology-Independent Loop-Free Alternate with Segment Routing for IS-IS*.

Supported and Unsupported Features for SRv6 Network Programming in IS-IS

SRv6 Network Programming in IS-IS Networks currently supports::

- Core IPv6 and dual stack, that is both IPv4 and IPv6 transport is supported.
- IPv4 and IPv6 payloads.
- Upto 6 SIDs in reduced mode at ingress router.
- Upto 7 SIDs in transit routers.

SRv6 Network Programming in IS-IS Networks currently does not support:

- Anycast for locator prefix.
- Shared Risk Link Group (SRLG) when computing backup paths.
- Static SRv6 tunnel with segment lists.
- ICMP error handling.
- SR-TE policy configuration for SRv6 Tunnel.
- Conflict resolution for Flexible Algorithm locators. Multiple nodes sharing the same locator prefix with different algorithm values could result in unexpected routing behavior.
- Interface group for End-X-SID.
- Configuring normal and extended admin-groups for IPv6 networks without MPLS, which is only allowed at [edit protocols mpls] hierarchy level.

SEE ALSO

> *srv6*
>
> *locator*
>
> *Example: Configuring SRv6 Network Programming in IS-IS Networks*
>
> *flex-algorithm*
>
> *definition*

WHAT'S NEXT

> For more information on SRv6 Network Programming, see the Example: Configuring SRv6 Network Programming in IS-IS Networks.

8

CHAPTER

NETCONF and Shell Sessions over Outbound HTTPS

How to Configure NETCONF or Shell Sessions over Outbound HTTPS

SUMMARY

Client applications can establish Network Configuration Protocol (NETCONF) sessions or shell sessions using outbound HTTPS on supported devices running Junos OS.

IN THIS SECTION

Understanding NETCONF and Shell Sessions over Outbound HTTPS

IN THIS SECTION

Benefits of NETCONF and Shell Sessions over Outbound HTTPS

- Enable NETCONF or shell client applications to manage devices that are not accessible through other protocols.
- Enable remote management of devices using certificate-based authentication for the outbound HTTPS client.

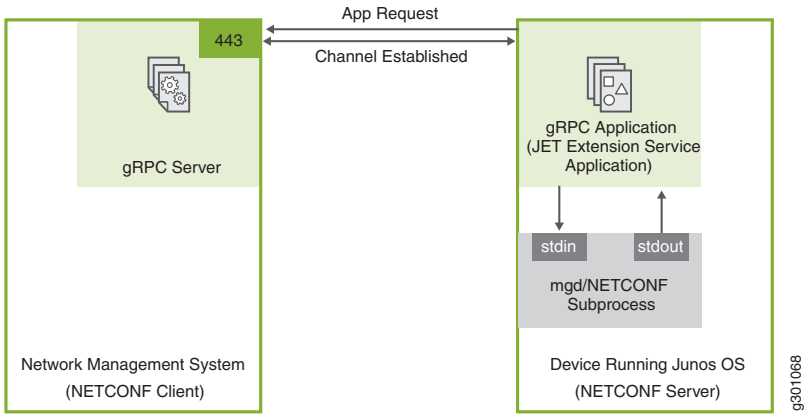
NETCONF and Shell Sessions over Outbound HTTPS Overview

You can establish NETCONF and shell sessions over outbound HTTPS between supported devices running Junos OS and a network management system. A NETCONF or shell session over outbound HTTPS enables

you to remotely manage devices that might not be accessible through other protocols such as SSH. This might happen, for example, if the device is behind a firewall, and the firewall or another security tool blocks those protocols. HTTPS, on the other hand, uses a standard port, which is typically allowed outbound in most environments.

The Junos OS with upgraded FreeBSD software image includes a Juniper Extension Toolkit (JET) application that supports establishing a NETCONF or shell session using outbound HTTPS. The JET application uses the gRPC framework to connect to the outbound HTTPS client, which consists of a gRPC server running on the network management system. gRPC is a language-agnostic, open-source remote procedure call (RPC) framework. Figure 6 on page 45 illustrates the outbound HTTPS setup in its simplest form.

Figure 6: NETCONF Session over Outbound HTTPS



In this scenario, the gRPC server acts as the NETCONF or shell client, and the JET application is the gRPC client and NETCONF or shell server. The gRPC server listens for connection requests on the specified port, which defaults to port 443. You configure the JET application as an extension service. The relevant connection and authentication information is passed to the script. While the script runs, it automatically attempts to connect to the gRPC server on the configured port.

The JET application and gRPC server establish a persistent HTTPS connection over a TLS-encrypted gRPC session. The JET application authenticates the gRPC server using an X.509 digital certificate, and if the authentication is successful, the requested NETCONF or shell session is established over this connection. The NETCONF operations and shell commands execute under the account privileges of the user configured for the extension service application.

The outbound HTTPS connection uses an X.509 digital certificate to authenticate the gRPC server. A digital certificate is an electronic means for verifying your identity through a trusted third party, known as a certificate authority or certification authority (CA). A certificate authority issues digital certificates, which can be used to establish a secure connection between two endpoints through certificate validation. The X.509 standard defines the format for the certificate. To establish a NETCONF or shell session over outbound HTTPS on supported devices running Junos OS, the gRPC server must have a valid X.509 certificate.

Table 2 on page 46 outlines the features supported for sessions over outbound HTTPS for a given Junos OS release. Devices running Junos OS Release 20.3 and later support multiple outbound HTTPS client connections; configuring one or more backup gRPC servers for a client; and establishing multiple, concurrent NETCONF and shell sessions between the outbound HTTPS client and device running Junos OS.

Table 2: Supported Features for Sessions over Outbound HTTPS

Feature/Component	Junos OS Release 20.2	Junos OS Release 20.3R1 or Later
Outbound HTTPS client connections	Support for connecting to a single outbound HTTPS client and configuring one gRPC server for that client. Connection details are configured as script arguments at the **[edit system extensions extension-service application file nc_grpc_app.py]** hierarchy level.	Support for connecting to multiple outbound HTTPS clients and configuring one or more backup gRPC servers for each client. Connection details are configured at the **[edit system services outbound-https]** hierarchy level.
Sessions	Supports a single NETCONF session.	Supports multiple, concurrent NETCONF and csh sessions for an outbound HTTPS client.
gRPC server certificate	Supports self-signed X.509 certificates only.	Supports self-signed or CA-signed X.509 certificates.
Authentication for the device running Junos OS	–	Supports configuring an identifier and shared secret to authenticate the device running Junos OS to the outbound HTTPS client.

Connection Workflow for Sessions over Outbound HTTPS

In a NETCONF or shell session over outbound HTTPS, the gRPC server running on the network management system acts as the NETCONF or shell client, and the JET application on the device running Junos OS is the gRPC client and NETCONF or shell server. Starting in Junos OS Release 20.3, you can configure multiple outbound HTTPS clients, and you can configure one or more backup gRPC servers for each client. The JET application connects to only one gRPC server in the client's server list at any one time. You can establish multiple, concurrent NETCONF and shell sessions with that gRPC server in releases that support multiple sessions.

The gRPC client and server perform the following actions to establish a NETCONF or shell session over outbound HTTPS:

1. The gRPC server listens for incoming connections on the specified port, or if no port is specified, on the default port 443.
2. The gRPC client initiates a TCP/IP connection with the configured gRPC server and port. If you configure an outbound HTTPS client with one or more backup gRPC servers, the gRPC client tries to connect to each server in the list until it establishes a connection.
3. The gRPC client sends a TLS ClientHello message to initiate the TLS handshake.
4. The gRPC server sends a ServerHello message and its certificate.
5. The gRPC client verifies the identity of the gRPC server.
6. On devices running Junos OS Release 20.3R1 or later, the gRPC client sends the device ID and shared secret configured for that outbound HTTPS client to the gRPC server.
7. The NETCONF or shell session is established as follows:
   - For connections to devices running Junos OS Release 20.2, a single NETCONF session is automatically established.
   - For connections to devices running Junos OS Release 20.3R1 or later, the outbound HTTPS client requests a NETCONF or shell session, and the gRPC server uses the device ID and shared secret to authenticate the device running Junos OS. If authentication is successful, the session is established.
8. If a NETCONF session is requested, the server and client exchange NETCONF **<hello>** messages.
9. The NETCONF or shell client application performs operations as needed.

If the device is running Junos OS Release 20.3R1 or later, the gRPC client initiates another TCP/IP connection with the same gRPC server, and the gRPC client and server repeat the process, which enables the outbound HTTPS client to establish multiple NETCONF and shell sessions with the device running Junos OS.

How to Establish NETCONF and Shell Sessions over Outbound HTTPS

You can use the JET application that is included as part of the Junos OS with upgraded FreeBSD software image to establish NETCONF and shell sessions over outbound HTTPS between network management systems (NMS) and supported devices running Junos OS. The JET application, configured as an extension service, initiates a connection to a gRPC server running on an NMS and establishes a persistent HTTPS connection over a TLS-encrypted gRPC session. The NETCONF or shell session runs over this HTTPS connection. In this scenario, the gRPC server is the NETCONF or shell client, and the JET application is the gRPC client and NETCONF or shell server.

The following hardware and software is required for establishing sessions over outbound HTTPS:

- Device running Junos OS with upgraded FreeBSD Release 20.2 or later that also supports running Juniper Extension Toolkit (JET) applications
- Network management system running Python 3.5 or later

Before the client and server can establish a NETCONF or shell session over outbound HTTPS, you must satisfy the requirements discussed in the following sections:

Obtain an X.509 Certificate for the gRPC Server

The outbound HTTPS connection uses an X.509 public key certificate to authenticate the identity of the gRPC server running on the network management system. The gRPC stack supports the X.509 v3 certificate format.

The requirements for the gRPC server's certificate are:

- To connect to devices running Junos OS Release 20.2, the certificate must be self-signed.

  To connect to devices running Junos OS Release 20.3 or later, the certificate can be self-signed or signed by a certificate authority (CA).
- The certificate must define either the gRPC server's hostname in the Common Name (CN) field, or it must define the gRPC server's IP address in the SubjectAltName (SAN) IP Address field. The device running Junos OS must use the same value to establish the connection to the server. If the SubjectAltName IP Address field is defined, the Common Name field is ignored during authentication.
- The certificate must be PEM-encoded and use a **.crt** extension.
- The certificate and its key must be named **server.crt** and **server.key**, respectively.

To use OpenSSL to obtain a certificate:

1. Generate a private key, and specify the key length in bits.

   `user@nms:~$` **openssl genrsa -out server.key 4096**

   ```
   Generating RSA private key, 4096 bit long modulus (2 primes)
   ...++++
   ..........................................................................++++
   e is 65537 (0x010001)
   ```

   NOTE: We recommend using 3072 bits or greater for the size of the private key.

2. If you are connecting to the gRPC server's IP address, update your **openssl.cnf** or equivalent configuration file to define the **subjectAltName=IP** extension with the address.

   `user@nms:~$` **cat openssl.cnf**

   ```
   # OpenSSL configuration file.
   ...
   extensions              = v3_sign
   ...
   [v3_sign]
   subjectAltName=IP:198.51.100.11
   ```

3. Generate a certificate signing request (CSR), which contains the client's public key and information about their identity.

   `user@nms:~$` **openssl req -new -key server.key -out server.csr**

   ```
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) [AU]:US
   State or Province Name (full name) [Some-State]:CA
   Locality Name (eg, city) []:Sunnyvale
   Organization Name (eg, company) [Internet Widgits Pty Ltd]: Juniper
   Organizational Unit Name (eg, section) []:
   Common Name (e.g. server FQDN or YOUR name) []:nms.example.com
   Email Address []:
   ```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

4. Generate the certificate by doing one of the following:

   - Send the CSR to a certificate authority to request an X.509 certificate, and provide the configuration file to include any additional extensions.
   - Sign the CSR with a CA to generate the client certificate, and include the **-extfile** option if you need to reference your configuration file and extensions.

     `user@nms:~$` **openssl x509 -req -in server.csr -CA RootCA.crt -CAkey RootCA.key -set_serial 0101 -out server.crt -days 365 -sha256 -extfile openssl.cnf**

     ```
     Signature ok
     subject=C = US, ST = CA, L = Sunnyvale, O = Juniper, CN = nms.example.com
     Getting Private key
     ```

   - Sign the CSR with the server key to generate a self-signed client certificate, and include the **-extfile** option if you need to reference your configuration file and extensions.

     `user@nms:~$` **openssl x509 -req -in server.csr -signkey server.key -out server.crt -days 365 -sha256 -extfile openssl.cnf**

     ```
     Signature ok
     subject=C = US, ST = CA, L = Sunnyvale, O = Juniper, CN = nms.example.com
     Getting Private key
     ```

5. Verify that the Common Name (CN) field and extensions, if provided, are correct.

   `user@nms:~$` **openssl x509 -text -noout -in server.crt**

   ```
   Certificate:
       Data:
           Version: 3 (0x2)
           ...
          Subject: C = US, ST = CA, L = Sunnyvale, O = Juniper, CN = nms.example.com

           ...
           X509v3 extensions:
               X509v3 Subject Alternative Name:
   ```

```
                IP Address:198.51.100.11
    ...
```

6. (Optional) To manage devices running Junos OS Release 20.2, copy the **server.crt** file to the **/var/db/scripts/jet** directory on the device running Junos OS to use the certificate file for authentication.

   `user@nms:~$` **scp server.crt <*device-hostname-or-ip*>:/var/db/scripts/jet**

```
Password:
server.crt
        100% 1862     3.9MB/s   00:00
```

> **NOTE:** You can omit this step if the key size is less than or equal to 4096 bits and you instead configure the certificate's contents in the JET application's **trusted_certs** argument on the device running Junos OS.

Set Up the gRPC Server

The network management system and the JET application on the device running Junos OS use the gRPC framework to establish a persistent HTTPS connection over a TLS-encrypted gRPC session. The network management system must have the gRPC stack installed and run a gRPC server that listens on the specified port for the connection request. Juniper Networks provides the necessary proto definition files and sample gRPC server application files in the Juniper Networks netconf-https-outbound repository on GitHub.

The network management system requires the following software:

- Python 3.5 or later

This section sets up the gRPC server on a network management system running Ubuntu 18.04. If you are running a different operating system, use the commands appropriate for your OS.

To set up the gRPC server on a network management system running Ubuntu 18.04:

1. Install **pip** for Python 3.

   `user@nms:~$` **sudo apt install python3-pip**

2. Install the **grpcio** package.

   `user@nms:~$` **sudo pip3 install grpcio==1.29.0**

3. Install the **grpcio-tools** package.

   ```
   user@nms:~$ sudo pip3 install grpcio-tools==1.18.0
   ```

4. Go to the Juniper GitHub repository at https://github.com/Juniper/netconf-https-outbound, and select the directory corresponding to the Junos OS release running on the managed devices.
   - For Junos OS Release 20.2, select the **20.2** directory.
   - For Junos OS Release 20.3 and later, select the **20.3** directory.
5. Download the application and proto files in the GitHub directory to the directory on the network management system where the gRPC server's certificate resides.
6. Use the protocol buffer compiler, **protoc**, to compile each proto definition file and generate Python code, which produces two output files for each proto file.

   ```
   user@nms:~$ python3 -m grpc_tools.protoc -I./ --python_out=. --grpc_python_out=. filename.proto
   ```

   For example, to compile the files to use with devices running Junos OS Release 20.3 or later, issue the following commands:

   ```
   user@nms:~$ python3 -m grpc_tools.protoc -I./ --python_out=. --grpc_python_out=.
   jnx_common_base_types.proto
   user@nms:~$ python3 -m grpc_tools.protoc -I./ --python_out=. --grpc_python_out=.
   jnx_netconf_service.proto
   user@nms:~$ ls jnx*.py
   jnx_common_base_types_pb2_grpc.py  jnx_netconf_service_pb2_grpc.py
   jnx_common_base_types_pb2.py       jnx_netconf_service_pb2.py
   ```

7. Start the gRPC server, and specify the port for the connection, if it's different from the default port 443.

   ```
   user@nms:~$ python3 nc_grpc_server.py -p 50051
   ```

   NOTE: You might need to execute the script with root permissions to listen on port 443.

The gRPC server listens indefinitely on the specified port for incoming connections. After you configure the device running Junos OS to connect to the gRPC server and a connection and session are established, you can perform NETCONF operations or shell commands as appropriate.

Configure the User Account for the NETCONF or Shell User

To establish a NETCONF or shell session over outbound HTTPS, you must first create a user account locally on the device running Junos OS. You use this account to perform the NETCONF or shell operations on the device for that session. The JET application runs using the permissions configured for this account.

To create a user account on a device running Junos OS:

1. Configure the **user** statement with a unique username, and include the **class** statement to specify a login class that has the permissions required for all actions to be performed by the user. For example:

   ```
   [edit system login]
   user@host# set user netconf-user class super-user
   ```

2. (Optional) Configure the **uid** and **full-name** statements to specify a unique user ID and the user's name.

   ```
   [edit system login]
   user@host# set user netconf-user uid 2001 full-name "NETCONF User"
   ```

3. Commit the configuration to activate the user account on the device.

   ```
   [edit system login]
   user@host# commit
   ```

4. Repeat the preceding steps on each device running Junos OS where the client needs to establish NETCONF or shell sessions over outbound HTTPS.

Configure the Outbound HTTPS Client (Junos OS Release 20.2)

In Junos OS Release 20.2, the JET application can only connect to a single outbound HTTPS client, and you configure the connection and authentication information for the client as command-line arguments to the JET script. Table 3 on page 53 outlines the arguments.

Table 3: nc_grpc_app.py Arguments

Argument	Value
--device or **-d**	The hostname or IPv4 address to which the JET application connects to the gRPC server. The argument value must match the hostname in the Common Name (CN) field or the IP address in the SubjectAltName IP address field in the gRPC server's certificate.

Table 3: nc_grpc_app.py Arguments *(continued)*

Argument	Value
--port or **-p**	(Optional) Port on which the JET application attempts to connect to gRPC server. Omit this argument to use the default port 443.
--trusted_certs or **-ts**	(Optional) The gRPC server's certificate contents between the **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----** lines, omitting any newlines. You can omit this argument if you instead copy the certificate to the **/var/db/scripts/jet** directory on the device. You must copy the certificate to the device for key sizes greater than 4096 bits.

Before you begin, you will need the values for the script arguments, including:

- The port on which the gRPC server is listening for connections.
- The contents of the SubjectAltName IP Address field, or if there is no such field, the contents of the Common Name (CN) field in the gRPC server's certificate.
- The contents of the gRPC server's certificate between **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----**, omitting any newlines. This information is only required when you configure the certificate contents as a script argument instead of copying the certificate to the device running Junos OS.

To configure the outbound HTTPS client:

1. Navigate to the hierarchy of the **nc_grpc_app.py** extension service application.

```
[edit]
user@host# edit system extensions extension-service application file nc_grpc_app.py
```

2. Configure the arguments that are passed to the application when it starts.

```
[edit system extensions extension-service application file nc_grpc_app.py]
user@host# set arguments "--device 198.51.100.11 --port 50051 --trusted_certs MIIFR***fhd7y"
```

3. Commit the configuration.

```
[edit system extensions extension-service application file nc_grpc_app.py]
user@host# commit
```

Configure the Outbound HTTPS Clients (Junos OS Release 20.3R1 and later)

Starting in Junos OS Release 20.3R1, you can configure multiple outbound HTTPS clients at the **[edit system services outbound-https]** hierarchy level, and you can configure multiple backup gRPC servers for each client. The JET application connects to only one gRPC server in the client's server list at any one time.

Before you configure the device, you will need the following information:

- The port on which the gRPC server is listening for connections.
- The contents of the SubjectAltName IP Address field, or if there is no such field, the contents of the Common Name (CN) field in the gRPC server's certificate.
- The contents of the gRPC server's certificate, if it's self-signed, or the contents of the CA certificates, if the server certificate is authenticated using a certificate chain.

To configure an outbound HTTPS client:

1. Navigate to the outbound HTTPS client hierarchy, and define an identifier that uniquely identifies the outbound HTTPS client.

```
[edit]
user@host# edit system services outbound-https client nms1
```

2. Define the device identifier, which is a user-defined string that the gRPC server uses to identify and authenticate the device running Junos OS during session establishment.

```
[edit system services outbound-https client nms1]
user@host# set device-id router1
```

3. Define a shared secret string, which is a user-defined string that the gRPC server uses to authenticate the device running Junos OS during session establishment.

```
[edit system services outbound-https client nms1]
user@host# set secret my-shared-secret
```

4. (Optional) Define the method used to reestablish a disconnected outbound HTTPS connection as **sticky** or **in-order**.

```
[edit system services outbound-https client nms1]
user@host# set reconnect-strategy sticky
```

5. (Optional) Define the time in seconds that the gRPC client waits in between attempts to connect to the outbound HTTPS client's list of servers.

```
[edit system services outbound-https client nms1]
user@host# set waittime 30
```

6. Configure the hostname or IPv4 address for one or more gRPC servers and the port on which the server is listening for outbound HTTPS connection requests.

   The hostname or IP address must match the value of the Common Name (CN) field or the SubjectAltName IP Address field, respectively, in that gRPC server's certificate.

```
[edit system services outbound-https client nms1]
user@host# set 198.51.100.11 port 50051
```

7. For each gRPC server, configure the **trusted_cert** statement with the certificate information required to authenticate the server.

   - If the server's certificate is self-signed, configure the contents of the gRPC server's certificate, omitting any newlines.

```
[edit system services outbound-https client nms1]
user@host# set 198.51.100.11 trusted-cert "-----BEGIN CERTIFICATE-----MIIFH***FjQ==-----END
  CERTIFICATE-----"
```

   - If the server's certificate is authenticated using a certificate chain, concatenate any intermediate CA and root CA certificates in that order, remove all newlines, and configure the resulting single string.

```
[edit system services outbound-https client nms1]
user@host# set 198.51.100.11 trusted-cert "-----BEGIN CERTIFICATE-----MIIFA***ioUS-----END
  CERTIFICATE----------BEGIN CERTIFICATE-----MIIFX***0xUc=-----END CERTIFICATE-----"
```

   NOTE: To easily generate the value for the **trusted_cert** statement, you can concatenate the appropriate certificates and remove any newlines, for example, by using a command similar to the following:

```
user@nms:~$ cat IntermediateCA.crt RootCA.crt | tr -d '\n' > allCA
```

8. Repeat the preceding steps for each outbound HTTPS client that will manage the device running Junos OS.

9. Commit the configuration.

```
[edit system services outbound-https client nms1]
user@host# commit and-quit
```

NOTE: If the outbound HTTPS extension service is already running, and you add, delete, or modify an outbound HTTPS client and commit the configuration, you do not need to restart the service for the changes to take effect. They are picked up automatically.

Configure the Extension Service for Outbound HTTPS on Devices Running Junos OS

On devices running Junos OS with upgraded FreeBSD that also support the Juniper Extension Toolkit (JET), the software image includes files that support NETCONF or shell sessions over outbound HTTPS. Table 4 on page 57 outlines the files, which are located in the **/var/db/scripts/jet** directory on supported devices.

Table 4: JET Files for Sessions over Outbound HTTPS

File Description	Junos OS Release 20.2R1	Junos OS Release 20.3R1 and Later
JET application that uses the gRPC framework to establish a persistent HTTPS connection with a gRPC server running on the network management system.	**nc_grpc_app.py**	**nc_grpc_app.pyc**
Required libraries	**nc_grpc_pb2.py** **nc_grpc_pb2_grpc.py**	**nc_grpc_app_lib.pyc**

To configure a device running Junos OS for sessions over outbound HTTPS:

1. Verify that the JET application and related files are present on the device.

   `user@host>` **file list /var/db/scripts/jet/nc***

   ```
   /var/db/scripts/jet/nc_grpc_app.pyc@ ->
   /packages/mnt/junos-runtime/var/db/scripts/jet/nc_grpc_app.pyc
   /var/db/scripts/jet/nc_grpc_app_lib.pyc@ ->
   /packages/mnt/junos-runtime/var/db/scripts/jet/nc_grpc_app_lib.pyc
   ```

2. Enter configuration mode.

   ```
   user@host> configure
   Entering configuration mode
   ```

3. Enable the device to run unsigned Python 3 applications.

   ```
   [edit]
   user@host# set system scripts language python3
   ```

4. On devices running Junos OS Release 20.3R1 or later, configure extension service notifications for the loopback address.

   ```
   [edit]
   user@host# set interfaces lo0 unit 0 family inet address 127.0.0.1
   user@host# set system commit notification configuration-diff-format xml
   user@host# set system services extension-service notification allow-clients address 127.0.0.1
   ```

5. Navigate to the hierarchy of the extension service application, and ensure that the filename, as defined in Table 4 on page 57, is correct for your release.

   ```
   [edit]
   user@host# edit system extensions extension-service application file nc_grpc_app.pyc
   ```

6. Configure the application to run as a daemonized process in the background.

   ```
   [edit system extensions extension-service application file nc_grpc_app.pyc]
   user@host# set daemonize
   ```

7. Configure the application to respawn on normal exit.

```
[edit system extensions extension-service application file nc_grpc_app.pyc]
user@host# set respawn-on-normal-exit
```

8. Configure the usename under whose privileges the application will execute and the NETCONF operations and shell commands will be performed.

```
[edit system extensions extension-service application file nc_grpc_app.pyc]
user@host# set username netconf-user
```

9. Commit the configuration.

```
[edit system extensions extension-service application file nc_grpc_app.pyc]
user@host# commit and-quit
```

When you commit the configuration, the **daemonize** option causes the application to start automatically.

10. Verify that the application is running.

```
user@host> show extension-service status nc_grpc_app.pyc
Extension service application details:
Name : nc_grpc_app
Process-id: 81383
Stack-Segment-Size: 16777216B
Data-Segment-Size: 134217728B
```

NOTE: If the application does not automatically start after you commit the configuration, issue the **show log jet.log** command, and review the log messages related to this application to troubleshoot the issue. After the application successfully starts, it logs messages to the **outbound_https.log** file.

Start the NETCONF or Shell Session

The gRPC server running on the network management system acts as the NETCONF or shell client, and the JET application on the device running Junos OS acts as the gRPC client and NETCONF or shell server. After you start the gRPC server and JET application, the JET application attempts to connect to the gRPC

server on the specified port. If the connection is successful, the gRPC client authenticates the gRPC server. If the server authentication is successful, the establishment of the NETCONF or shell session depends on the Junos OS release.

- When managing devices running Junos OS Release 20.2, the NETCONF session starts automatically.
- When managing devices running Junos OS Release 20.3R1 or later, you must request the session and specify the device ID, shared secret, and session type.

Before you begin, you will need the following information to manage devices running Junos OS Release 20.3 or later:

- The device identifier and shared secret string configured for that outbound HTTPS client

To establish a NETCONF or shell session over outbound HTTPS:

1. On the network management system, if you did not already start the gPRC server, start the server, and specify the port for the connection. The script output varies for different releases.

   `user@nms:~$` **python3 nc_grpc_server.py -p 50051**

   ```
   2020-08-03 13:45:52,278 [INFO ]  /home/user/
   2020-08-03 13:45:52,279 [INFO ]  first parent process is exited
   2020-08-03 13:45:52,287 [INFO ]  second parent process is exited
   ```

   When managing devices running Junos OS Release 20.2, the NETCONF session starts automatically.

2. To establish one or more sessions with a device running Junos OS Release 20.3R1 or later, execute the **request_session.py** script, and specify the session type as well as the device ID and shared secret that you configured for that outbound HTTPS client on the device running Junos OS. For example:

   - To request a csh session, which is the default, you do not need to specify a session type.

     `user@nms:~$` **python3 request_session.py -d router1 -sk my-shared-secret**

   - To request a NETCONF session, include the **-s netconf** option.

     `user@nms:~$` **python3 request_session.py -d router1 -sk my-shared-secret -s netconf**

   If the authentication of the device running Junos OS is successful, the requested session starts.

3. Verify that the session is successfully established by reviewing the session output, which should include NETCONF capabilities for NETCONF sessions or the **csh session is started** output for shell sessions.

   ```
   Initial hand shake completed and the client is trusted
   <!-- No zombies were killed during the creation of this user interface -->
   ```

```
<!-- user netconf-user, class j-super-user -->
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    ...
    <capability>http://xml.juniper.net/netconf/junos/1.0</capability>
    <capability>http://xml.juniper.net/dmi/system/1.0</capability>
  </capabilities>
  <session-id>57602</session-id>
</hello>
]]>]]>
```

4. Perform NETCONF or shell operations as necessary.

   <rpc><get-configuration/></rpc>

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:junos="http://xml.juniper.net/junos/20.2R1/junos">
<configuration xmlns="http://xml.juniper.net/xnm/1.1/xnm"
junos:changed-seconds="1592517292" junos:changed-localtime="2020-06-18 14:54:52
 PDT">
...
</configuration>
</rpc-reply>
]]>]]>
```

5. When you are finished with the session, type **Ctrl+C**.

```
^CForce exit
Killed
```

6. When you are finished using the outbound HTTPS connection, you can stop the extension service application on the device running Junos OS by deleting or deactivating the relevant hierarchy in the configuration and then committing the change.

   user@host# **delete system extensions extension-service application file nc_grpc_app.py**
   user@host# **commit and-quit**

WHAT'S NEXT

> For more information about NETCONF sessions on devices running Junos OS, see the NETCONF XML Management Protocol Developer Guide.