Junos OS

Junos Telemetry Interface Feature Guide

Release
16.1R3

Modified: 2017-01-25

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

Junos OS Junos Telemetry Interface Feature Guide
Release 16.1R3

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 http://www.juniper.net/support/eula.html. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

List of Figures

List of Tables

About the Documentation

Documentation and Release Notes

To obtain the most current version of all Juniper Networks® technical documentation, see the product documentation page on the Juniper Networks website at http://www.juniper.net/techpubs/.

If the information in the latest release notes differs from the information in the documentation, follow the product Release Notes.

Juniper Networks Books publishes books by Juniper Networks engineers and subject matter experts. These books go beyond the technical documentation to explore the nuances of network architecture, deployment, and administration. The current list can be viewed at http://www.juniper.net/books.

Supported Platforms

For the features described in this document, the following platforms are supported:

- MX Series
- PTX Series

Using the Examples in This Manual

If you want to use the examples in this manual, you can use the **load merge** or the **load merge relative** command. These commands cause the software to merge the incoming configuration into the current candidate configuration. The example does not become active until you commit the candidate configuration.

If the example configuration contains the top level of the hierarchy (or multiple hierarchies), the example is a *full example*. In this case, use the **load merge** command.

If the example configuration does not start at the top level of the hierarchy, the example is a *snippet*. In this case, use the **load merge relative** command. These procedures are described in the following sections.

Merging a Full Example

To merge a full example, follow these steps:

1. From the HTML or PDF version of the manual, copy a configuration example into a text file, save the file with a name, and copy the file to a directory on your routing platform.

   For example, copy the following configuration to a file and name the file **ex-script.conf**. Copy the **ex-script.conf** file to the **/var/tmp** directory on your routing platform.

   ```
   system {
     scripts {
       commit {
         file ex-script.xsl;
       }
     }
   }
   interfaces {
     fxp0 {
       disable;
       unit 0 {
         family inet {
           address 10.0.0.1/24;
         }
       }
     }
   }
   ```

2. Merge the contents of the file into your routing platform configuration by issuing the **load merge** configuration mode command:

   ```
   [edit]
   user@host# load merge /var/tmp/ex-script.conf
   load complete
   ```

Merging a Snippet

To merge a snippet, follow these steps:

1. From the HTML or PDF version of the manual, copy a configuration snippet into a text file, save the file with a name, and copy the file to a directory on your routing platform.

   For example, copy the following snippet to a file and name the file **ex-script-snippet.conf**. Copy the **ex-script-snippet.conf** file to the **/var/tmp** directory on your routing platform.

   ```
   commit {
     file ex-script-snippet.xsl; }
   ```

2. Move to the hierarchy level that is relevant for this snippet by issuing the following configuration mode command:

```
[edit]
user@host# edit system scripts
[edit system scripts]
```

3. Merge the contents of the file into your routing platform configuration by issuing the **load merge relative** configuration mode command:

```
[edit system scripts]
user@host# load merge relative /var/tmp/ex-script-snippet.conf
load complete
```

For more information about the **load** command, see CLI Explorer.

Documentation Conventions

Table 1 on page xi defines notice icons used in this guide.

Table 1: Notice Icons

Icon	Meaning	Description
	Informational note	Indicates important features or instructions.
	Caution	Indicates a situation that might result in loss of data or hardware damage.
	Warning	Alerts you to the risk of personal injury or death.
	Laser warning	Alerts you to the risk of personal injury from a laser.
	Tip	Indicates helpful information.
	Best practice	Alerts you to a recommended use or implementation.

Table 2 on page xi defines the text and syntax conventions used in this guide.

Table 2: Text and Syntax Conventions

Convention	Description	Examples
Bold text like this	Represents text that you type.	To enter configuration mode, type the **configure** command: user@host> **configure**

Table 2: Text and Syntax Conventions *(continued)*

Convention	Description	Examples
`Fixed-width text like this`	Represents output that appears on the terminal screen.	`user@host>` **show chassis alarms** `No alarms currently active`
Italic text like this	• Introduces or emphasizes important new terms. • Identifies guide names. • Identifies RFC and Internet draft titles.	• A policy *term* is a named structure that defines match conditions and actions. • *Junos OS CLI User Guide* • RFC 1997, *BGP Communities Attribute*
Italic text like this	Represents variables (options for which you substitute a value) in commands or configuration statements.	Configure the machine's domain name: [edit] root@# **set system domain-name** *domain-name*
Text like this	Represents names of configuration statements, commands, files, and directories; configuration hierarchy levels; or labels on routing platform components.	• To configure a stub area, include the **stub** statement at the **[edit protocols ospf area area-id]** hierarchy level. • The console port is labeled **CONSOLE**.
< > (angle brackets)	Encloses optional keywords or variables.	**stub <default-metric** *metric***>;**
| (pipe symbol)	Indicates a choice between the mutually exclusive keywords or variables on either side of the symbol. The set of choices is often enclosed in parentheses for clarity.	**broadcast | multicast** **(*string1* | *string2* | *string3*)**
# (pound sign)	Indicates a comment specified on the same line as the configuration statement to which it applies.	**rsvp { # Required for dynamic MPLS only**
[ ] (square brackets)	Encloses a variable for which you can substitute one or more values.	**community name members [** ***community-ids*** **]**

Indention and braces ({ })	Identifies a level in the configuration hierarchy.	[edit]
		routing-options {
		  static {
		  route default {
		  nexthop *address*;
		  retain;
		  }
		  }
		}

; (semicolon)	Identifies a leaf statement at a configuration hierarchy level.	
GUI Conventions		
Bold text like this	Represents graphical user interface (GUI) items you click or select.	• In the Logical Interfaces box, select **All Interfaces**. • To cancel the configuration, click **Cancel**.

Table 2: Text and Syntax Conventions *(continued)*

Convention	Description	Examples
> (bold right angle bracket)	Separates levels in a hierarchy of menu selections.	In the configuration editor hierarchy, select **Protocols>Ospf**.

Documentation Feedback

We encourage you to provide feedback, comments, and suggestions so that we can improve the documentation. You can provide feedback by using either of the following methods:

- Online feedback rating system—On any page of the Juniper Networks TechLibrary site at http://www.juniper.net/techpubs/index.html, simply click the stars to rate the content, and use the pop-up form to provide us with information about your experience. Alternately, you can use the online feedback form at http://www.juniper.net/techpubs/feedback/.
- E-mail—Send your comments to **techpubs-comments@juniper.net**. Include the document or topic name, URL or page number, and software version (if applicable).

Requesting Technical Support

Technical product support is available through the Juniper Networks Technical Assistance Center (JTAC). If you are a customer with an active J-Care or Partner Support Service support contract, or are covered under warranty, and need post-sales technical support, you can access our tools and resources online or open a case with JTAC.

- JTAC policies—For a complete understanding of our JTAC procedures and policies, review the *JTAC User Guide* located at http://www.juniper.net/us/en/local/pdf/resource-guides/7100059-en.pdf.
- Product warranties—For product warranty information, visit http://www.juniper.net/support/warranty/.
- JTAC hours of operation—The JTAC centers have resources available 24 hours a day, 7 days a week, 365 days a year.

Self-Help Online Tools and Resources

For quick and easy problem resolution, Juniper Networks has designed an online self-service portal called the Customer Support Center (CSC) that provides you with the following features:

- Find CSC offerings: http://www.juniper.net/customers/support/
- Search for known bugs: http://www2.juniper.net/kb/
- Find product documentation: http://www.juniper.net/techpubs/
- Find solutions and answer questions using our Knowledge Base: http://kb.juniper.net/

- Download the latest versions of software and review release notes: http://www.juniper.net/customers/csc/software/
- Search technical bulletins for relevant hardware and software notifications: http://kb.juniper.net/InfoCenter/
- Join and participate in the Juniper Networks Community Forum: http://www.juniper.net/company/communities/
- Open a case online in the CSC Case Management tool: http://www.juniper.net/cm/

To verify service entitlement by product serial number, use our Serial Number Entitlement (SNE) Tool: https://tools.juniper.net/SerialNumberEntitlementSearch/

Opening a Case with JTAC

You can open a case with JTAC on the Web or by telephone.

- Use the Case Management tool in the CSC at http://www.juniper.net/cm/.
- Call 1-888-314-JTAC (1-888-314-5822 toll-free in the USA, Canada, and Mexico).

For international or direct-dial options in countries without toll-free numbers, see http://www.juniper.net/support/requesting-support.html.

PART 1

Junos Telemetry Interface Overview

CHAPTER 1

Overview of the Junos Telemetry Interface

Overview of the Junos Telemetry Interface

As the number of objects on the network and the metrics they generate have grown, the traditional models, such as SNMP, used to gather operational statistics for monitoring the health of a network, have imposed limits on network element scale and efficiency. The so-called pull model used by SNMP and the CLI, which requires additional processing to periodically poll the network element, directly limits scaling.

The Junos Telemetry Interface (JTI) overcomes these limits by relying on a so-called push model to deliver data asynchronously, which eliminates polling. A request to send data is sent once by a management station to stream periodic updates. As a result, JTI is highly scalable and can support the monitoring of thousands of objects in a network.

NOTE: Junos Telemetry Interface was introduced in Junos OS Release 15.1F3, on MX Series routers with interfaces configured on MPC1 through MPC6E, and on PTX Series routers with interfaces configured on FPC3. Starting in Junos OS Release 15.1F5, Junos Telemetry Interface is also supported on MPC7E, MPC8E, and MPC9E on MX Series routers.

Starting with Junos OS Release 16.1R3, FPC1, FPC2, and dual Routing Engines on PTX Series routers are also supported.

Telemetry Sensors and Data Models

The Junos Telemetry Interface enables you to provision sensors to collect and export data for various system resources, such as physical interfaces and firewall filters. Two data models, each of which uses a different mode of transport, are supported:

- An open and extensible data model defined by Juniper Networks. Data is generated as Google protocol buffers (gpb) structured messages. The files that define each **.proto** message are published on the Juniper Networks web site. Native sensors export data close to the source, such as the line card or network processing unit (NPU), using the

User Datagram Protocol (UDP). Because this model features a distributed architecture, it scales easily.

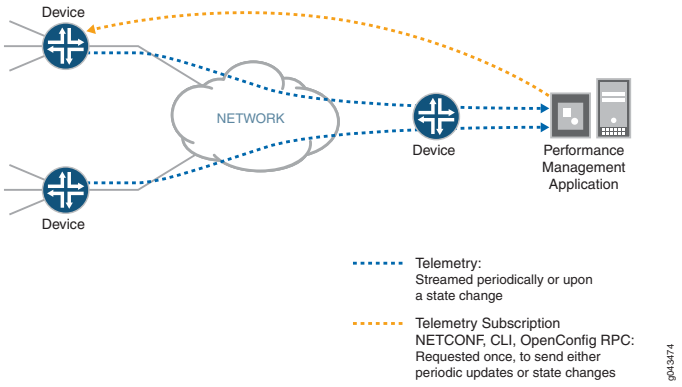
- An OpenConfig data model that generates data as gpb messages in a universal key/value format. OpenConfig for Junos OS, which you must download, supports the YANG data models. gRPC remote procedure calls (gRPC) are used to provision sensors and to subscribe to and receive telemetry data. gRPC is based on TCP, and supports SSL encryption, so it is considered secure and reliable. If your Juniper Networks device is running a version of Junos OS with the upgraded FreeBSD kernel, this model requires you to download the Network Agent package, which runs on the Routing Engine and provides interfaces to manage gRPC subscriptions. For other versions of Junos OS, Network Agent functionality is embedded in the software.

Uses and Benefits

A primary function of the Junos Telemetry Interface is performance monitoring. Streaming data to a performance management system enables network administrators to measure trends in link and node utilization, and troubleshoot such issues as network congestion in real time, for example.

In a typical deployment, the network element, or device, streams duplicate data to two destination servers that function as performance management system collectors. Streaming data to two collectors provides redundancy. See Figure 1 on page 4 for an illustration of how the performance management system collectors request data and how the device streams data. The device provisions sensors to collect and export data using command-line interface (CLI), configuration through NETCONF, or gRPC subscription calls. The collectors request data by initiating a telemetry subscription. Data is requested only once and is streamed periodically.

Figure 1: Telemetry Streaming for Performance Management



Other applications of the Junos Telemetry Interface include providing real-time data to support operational state synchronization between a network element and an external controller, such as the Northstar Controller, which automates the creation of traffic-engineering paths across the network. The NorthStar Controller can subscribe to telemetry data about certain network elements, such as label-switched path (LSP) statistics.

Release History Table

Release	Description
16.1R3	Starting with Junos OS Release 16.1R3, FPC1, FPC2, and dual Routing Engines on PTX Series routers are also supported.
15.1F5	Starting in Junos OS Release 15.1F5, Junos Telemetry Interface is also supported on MPC7E, MPC8E, and MPC9E on MX Series routers.
15.1F3	Junos Telemetry Interface was introduced in Junos OS Release 15.1F3, on MX Series routers with interfaces configured on MPC1 through MPC6E, and on PTX Series routers with interfaces configured on FPC3.

Related Documentation

- Understanding the Junos Telemetry Interface Export Format of Collected Data on page 10
- Understanding OpenConfig and gRPC on Junos Telemetry Interface on page 51

PART 2

Configuring Native Sensors

CHAPTER 2

Export Format of Collected Data

Understanding the Junos Telemetry Interface Export Format of Collected Data

The Junos Telemetry Interface supports two ways of exporting data in the Google protocol buffers (gpb) format:

- Through UDP from so-called native sensors that export data close to the source, such as the line card or network processing unit (NPU). Juniper Networks defines the data model, which is open and extensible.
- Through gRPC remote procedure calls (gRPC) that export data through the Routing Engine. The data model is defined by OpenConfig, which supports the use of vendor-neutral data models to configure and manage the network. OpenConfig for Junos OS supports the YANG data models. For platforms that are running a version of Junos OS based on an upgraded FreeBSD kernel only, you must install a separate package called Network Agent that functions as a gRPC server and terminates the RPC interfaces. For all other versions of Junos OS, the Network Agent functionality is embedded in the software.

This section describes the format of data exported from native senors using UDP. The data is encapsulated into a UDP header, which is in turn encapsulated in the IPv4 payload. This model of the Junos Telemetry Interface is based a distributed architecture, through which the data generated by configured sensors is exported directly from the data plane, bypassing the control plane, and thus conserving these resources to perform other necessary functions.

NOTE: The Junos Telemetry Interface was introduced in Junos OS Release 15.1F3, on MX Series routers with interfaces configured on MPC1 through MPC6E, and on PTX Series routers with interfaces configured on FPC3. Starting in Junos OS Release 15.1F6, Junos Telemetry Interface is also supported on MPC7E, MPC8E, and MPC9E on MX Series routers.

Starting with Junos OS Release 16.1R3, FPC1, FPC2, and dual Routing Engines on PTX Series routers are also supported.

- Understanding the Sensor Data Encapsulation Format on page 10

Understanding the Sensor Data Encapsulation Format

A native sensor exports data close to the source using UDP. Various types of telemetry data, such as physical interface statistics, firewall filter counter statistics, or statistics for label-switched paths (LSPs) can be exported. A sensor starts to emit data as soon as it is enabled.

The sensor data is represented as a single structured Google protocol buffers message, named **TelemetryStream**. The message, or **.proto** file, shown below, includes several attributes that identify the data source, such as a line card, a Packet Forwarding Engine, or a Routing Engine. The name of the configured sensor is also included. For more information about how to configure sensors, see "Configuring a Junos Telemetry Interface Sensor (CLI Procedure)" on page 15 For a a list of supported native sensors, see **sensor**.

You must also download the **.proto** files for all the sensors supported to a streaming server or collector. From a Web browser, navigate to the All Junos Platforms software download URL on the Juniper Networks page: http://www.juniper.net/support/downloads/. After you select the name of the Junos OS platform and the release number, go to the **Tools** section and download the **Junos Telemetry Interface Data Model Files** package. For more information about configuring a streaming-server, see streaming-server (Junos Telemetry Interface).

Google protocol buffers message Definition

Following is the message definition for **TelemetryStream** in the Google Protocol Buffers definition language. It shows several optional nested structures, such as **EnterpriseSensors**, which carry privately defined sensor data.

```
//
// This file defines the top level message used for all Juniper
// Telemetry packets encoded to the protocol buffer format.
// The top level message is TelemetryStream.
//

import "google/protobuf/descriptor.proto";

extend google.protobuf.FieldOptions {
    optional TelemetryFieldOptions telemetry_options = 1024;
}

message TelemetryFieldOptions {
    optional bool is_key             = 1;
    optional bool is_timestamp       = 2;
    optional bool is_counter         = 3;
    optional bool is_gauge           = 4;
}

message TelemetryStream {
    // router name or export IP address
    required string system_id        = 1 [(telemetry_options).is_key = true];

    // line card / RE (slot number)
    optional uint32 component_id     = 2 [(telemetry_options).is_key = true];

    // PFE (if applicable)
    optional uint32 sub_component_id = 3 [(telemetry_options).is_key = true];

    // configured sensor name
    optional string sensor_name      = 4 [(telemetry_options).is_key = true];

    // sequence number, monotonically increasesing for each
    // system_id, component_id, sub_component_id + sensor_name.
    optional uint32 sequence_number  = 5;

    // timestamp (milliseconds since 00:00:00 UTC 1/1/1970)
    optional uint64 timestamp        = 6 [(telemetry_options).is_timestamp =
true];

    // major version
    optional uint32 version_major    = 7;

    // minor version
    optional uint32 version_minor    = 8;
```

```
    optional IETFSensors ietf        = 100;

    optional EnterpriseSensors enterprise = 101;
}

message IETFSensors {
    extensions 1 to max;
}

message EnterpriseSensors {
    extensions 1 to max;
}

extend EnterpriseSensors {
    // re-use IANA assigned numbers
    optional JuniperNetworksSensors juniperNetworks = 2636;
}

message JuniperNetworksSensors {
    extensions 1 to max;
}
```

The **TelemetryStream** message also includes optional nested structures that carry different types of data. One structure carries enterprise, that is, privately defined data. Individual companies, such as Juniper Networks, define and maintain the attributes generated by enterprise sensors. Each company is assigned a unique attribute identifier. The current convention is to use IANA-assigned enterprise MIB identifiers for each attribute. For Juniper Networks, this assigned identifier is 2636.

BEST PRACTICE: To verify that a particular message type has been exported and received, check for those attributes under **TelemetryStream.enterprise.juniperNetworks** in the gpb message.

See Table 3 on page 12 for descriptions of each element collected by sensor data, including semantics and corresponding schema.

Table 3: Individual Data Element Types in the gpb Message

Element Type	Description
Counter	An unsigned integer that increases monotonically. When it reaches its maximum value, it starts back at zero.
Gauge	An unsigned 32-bit or 64-bit integer that can increase or decrease in value. An example of the data represented by this element is the instantaneous value of a specific resource, such as queue depth or temperature.
Rate	Rate at which a base metric changes, such as a counter or a gauge. For this element type, units of measurement are defined explicitly (such as bits per second), as well the interval over which the rate is collected.

Table 3: Individual Data Element Types in the gpb Message *(continued)*

Element Type	Description
Average	The average of several samples of a base metric. For example, an *average queue depth* data element would be calculated by averaging several elements of the queue depth. For this element type, we strongly recommend defining the number of measurements used to compute the average, as well as the time interval between the measurements. Otherwise, you should define explicitly the means by which this average value is calculated.
Peak	Maximum value among several samples of a base metric. For example, a *peak queue depth* element would be calculated by comparing several measurements of the queue depth and selecting the maximum. For this data element type, we strongly recommend that you define the number of measurements used to compute the peak value, as well as the time interval between measurements. Otherwise, define explicitly how this peak value is defined. You must also know whether this value is never cleared and thus represents the overall maximum value over all time.

NOTE: Each data element type also includes element subsets. For example, the data elements Counter and Gauge would include subsets for rate, average, and peak measurements.

Related Documentation

- Decoding Junos Telemetry Interface Data With UNIX Utilities on page 39

CHAPTER 3

Configuring Junos Telemetry Interface (CLI Procedure)

Configuring a Junos Telemetry Interface Sensor (CLI Procedure)

Junos Telemetry Interface provides for the highly scalable streaming of telemetry information. Unlike previous monitoring systems, such as SNMP, which use the so-called pull model, the Junos Telemetry Interface uses the push model to collect data. The push model overcomes earlier scaling limits and reduces the processing required by the management station. You can enable monitoring and streaming of data for various system resources, such as physical and logical interfaces and firewall filters. To monitor a specific system resource, you configure a sensor. Each sensor configuration requires three main components:

- Sensor profile—Enables the system resource to monitor and allows you to set related parameters, such as the destination server to send data.
- Export profile—Specifies the attributes for the process of exporting collected data, such as the transport protocol to use and the interval at which to collect data.
- Streaming server profile—Specifies the server for collecting data and related parameters, including the destination IP address and port number.

NOTE: Junos Telemetry Interface was introduced in Junos OS Release 15.1F3 on MX Series routers with interfaces configured on MPC1 through MPC6E and on PTX Series routers with interfaces configured on FPC3. Starting in Junos OS Release 15.1F5, Junos Telemetry Interface is also supported on MPC7E, MPC8E, and MPC9E on MX Series routers.

Starting with Junos OS Release 16.1R3, FPC1 and FPC2 on PTX Series routers are also supported.

BEST PRACTICE: We recommend that you configure at least one export profile and at least one streaming server before you configure a sensor profile.

This way you can associate an export profile and a streaming server with the sensor profile configuration.

Before you begin:

- Configure a connection from your Juniper Networks device to a server that is using in-band management interfaces.

- Configuring an Export Profile on page 16
- Configuring a Streaming Server Profile on page 18
- Configuring a Sensor Profile on page 18
- Verifying Junos Telemetry Interface Sensor Configuration on page 20

Configuring an Export Profile

An export profile defines the parameters of the export process of data generated through the Junos Telemetry Interface. You must configure at least one export profile, but you can configure multiple export profiles. Each export profile can be associated with multiple sensor profiles. However, you can associate only one export profile with a specific sensor profile.

To configure an export profile:

1. Specify a name for the export profile.

   ```
   [edit services analytics]
   user@host# set export-profile name]
   ```

   For example, to specify an export-profile name of **export-params**:

   ```
   [edit services analytics]
   user@host# set export-profile export-params
   ```

2. Specify the source IP address of exported packets.

   ```
   [edit services analytics export-profile name]
   user@host# set local-address ip-address
   ```

   For example, to specify a source IP address of 192.0.2.3 for an export profile with the name **export-params**:

   ```
   [edit services analytics export-profile export-params]
   user@host# set local-address 192.0.2.3
   ```

3. Specify the source port number of exported packets.

   ```
   [edit services analytics export-profile name]
   user@host# set local-port number
   ```

   For example, to specify a source port number of 21111 for an export profile with the name **export-params**:

   ```
   [edit services analytics export-profile export-params]
   user@host# set local-port 21111
   ```

4. Specify the interval, in seconds, at which the sensor generates telemetry data.

```
[edit services analytics export-profile name]
user@host# set reporting-rate seconds
```

For example, to specify an interval of 20 seconds at which any sensor associated with the export-profile with the name **export-params** generates telemetry data :

```
[edit services analytics sensor export-profile export-params]
user@host# set reporting-rate 20
```

5. Specify the format to define the structure of the exported data.

NOTE: The only currently supported format is Google protocol buffers (gpb)

```
[edit services analytics export-profile name]
user@host# set format gpb
```

For example, to specify the Google protocol buffers format for exported data for an export-profile with the name **export-params**:

```
[edit services analytics export-profile export-params]
user@host# set format gpb
```

6. Specify the transport protocol to carry the telemetry data in the IP packets.

```
[edit services analytics export-profile name]
user@host# set transport protocol-name
```

For example, to specify the UDP as the transport protocol for telemetry data for an export profile with the name **export-params**:

```
[edit services analytics export-profile export-params]
user@host# set transport udp
```

7. (Optional) Specify the DiffServ code point (DSCP) value to assign to exported packets.

NOTE: The default value is 0 (zero).

```
[edit services analytics export-profile name]
user@host# set dscp value
```

For example, to specify a DSCP value of 20 for an export profile with the name **export-params**:

```
[edit services analytics export-profile export-params]
user@host# set dscp 20
```

8. (Optional) Specify a forwarding class to assign to exported packets.

NOTE: You can specify a forwarding class only for packets exported by Packet Forwarding Engine sensors. The default value is best-effort.

```
[edit services analytics export-profile name]
user@host# set forwarding-class class-name
```

For example, to specify a forwarding class of **assured-forwarding** for an export-profile with the name **export-params**:

```
[edit services analytics export-profile name]
user@host# set forwarding-class assured forwarding
```

Configuring a Streaming Server Profile

A server profile defines the parameters of the server that collects exported telemetry data. You can define more than one server profile. You can also associate the same server profile with more than one sensor profile. Starting in Junos OS Release 15.1F6, you can associate more than one server with a specific sensor.

To define the profile of a streaming server to collect exported telemetry data:

1. Specify the name of the streaming sever.

   ```
   [edit services analytics]
   user@host# set streaming-server server-name
   ```

   For example, to specify a streaming-server name of **telemetry server**:

   ```
   [edit services analytics]
   user@host# set streaming-server telemetry-server
   ```

2. Specify a destination IP address for the exported packets.

   ```
   [edit services analytics streaming-server server-name]
   user@host# set remote-address ip-address
   ```

   For example, to specify a destination address of 192.0.2.2 for a streaming server with the name **telemetry-server**:

   ```
   [edit services analytics streaming-server telemetry-server]
   user@host# set remote-address 192.0.2.2
   ```

3. Specify a destination port number for the exported packets.

   ```
   [edit services analytics streaming-server server-name]
   user@host# set remote-port number
   ```

   For example, to specify a destination port number of 30000 for a streaming server with the name **telemetry-server**:

   ```
   [edit services analytics streaming-server telemetry-server]
   user@host# set remote-port 30000
   ```

Configuring a Sensor Profile

A sensor profile defines the parameters of the system resource to monitor and stream data. You can enable only one system resource to monitor for each sensor profile. Configure a different sensor profile for each system resource you want to monitor. You can, however, configure more than one sensor to monitor the same system resource. For example, you might want to configure different parameters for exporting data for the same system resource.

To configure a sensor profile:

1. Specify the name of the sensor.

   ```
   [edit services analytics]
   user@host# set sensor sensor-name
   ```

   For example, to specify a sensor name of **interface-1**:

   ```
   [edit services analytics]
   user@host# set sensor interface-1
   ```

2. Specify the system resource to monitor and stream data.

   ```
   [edit services analytics sensor sensor-name]
   user@host# set resource resource-string-identifier
   ```

   For example, to enable monitoring of logical interfaces for sensor **interface-1**:

   ```
   [edit services analytics sensor interface-1]
   user@host# set resource /junos/system/linecard/interface/logical/usage/
   ```

   NOTE: You must enter the resource string exactly.

3. (Optional) Specify a regular expression to filter data for the system resource you specified in Step 2. If you do not specify a regular expression, the system resource is monitored globally, that is, systemwide.

   ```
   [edit services analytics sensor sensor-name]
   user@host# set resource-filter regular-expression
   ```

   For example, to filter data only for Ethernet logical interfaces for sensor **interface-1**:

   ```
   [edit services analytics sensor interface-1]
   user@host# set resource-filter et-*
   ```

4. Specify the name of a export profile configured at the **[edit export-profile *profile-name*]** hierarchy level to associate with the sensor profile. This export profile defines the parameters for exporting telemetry data.

   ```
   [edit services analytics sensor sensor-name]
   user@host# set export-name export-profile-name
   ```

   For example, to associate an export profile named **export-params** with a sensor named **interface-1**:

   ```
   [edit services analytics sensor interface-1]
   user@host# set export-name export-params
   ```

5. Specify the name of a streaming server name configured at the **[edit services analytics streaming-server *server-name*]** hierarchy level to collect exported data.

   NOTE: Starting in Junos OS Release 15.1F6, you can specify more than one streaming server for a sensor profile. To specify more than one streaming server for a sensor, you must enclose the names in brackets.

   ```
   [edit services analytics sensor sensor-name]
   ```

```
user@host# set streaming-server server-name
```

For example, to associate a streaming server name **telemetry-server** with a sensor named **interface-1**:

```
[edt services analytics sensor interface-1]
user@host# set streaming-server telemetry-server
```

Verifying Junos Telemetry Interface Sensor Configuration

Purpose Confirm your configuration.

Action From configuration mode, confirm your configuration by entering the **show services analytics** command. If your output does not display the intended configuration, repeat the instructions in this configuration procedure to correct the configuration.

```
user@host# show services analytics
streaming-server telemetry-server {
   remote-address 192.-0.2..2;
   remote-port 30000;
}
export-profile export-params {
   local-address 192.0.2.3;
   local-port 21111;
   dscp 20;
   forwarding-class assured-forwarding;
   reporting-rate 20;
   format gpb;
   transport udp;
}
sensor interface-1 {
   server-name telemetry-server;
   export-name export-params;
   resource /junos/system/linecard/interface/logical/usage/;
   resource-filter et-*;
}
```

After you commit the configuration, verify that the sensor is enabled by issuing the **show agent sensors** operational command.

```
user@host> show agent sensors

Sensor Information :

   Name                                : interface-1
   Resource                            : /junos/system/linecard/interface/logical/usage/
   Version                             : 1.0
   Sensor-id                           : 193570469
   Resource-filter                     : et-*

   Server Information :

       Name                            : telemetry-server
       Scope-id                        : 0
       Remote-Address                  : 192.0.0.2
       Remote-port                     : 30000

   Profile Information :
```

```
Name                            : export-params
Rep-interval                    : 300
Address                         : 192.0.0.3
Port                            : 21111
Timestamp                       : 1
Format                          : GPB
Transport                       : UDP
DSCP                            : 20
Forwarding-class                : assured-forwarding
```

CHAPTER 4

Junos Telemetry Interface Configuration Statements and Operational Commands

export-profile (Junos Telemetry Interface)

Syntax

```
export-profile name {
  dscp value;
  format file-format;
  forwarding-class (assured-forwarding | best-effort | expedited-forwarding |
    network-control);
  local-address ip-address;
  local-port source-port-number;
  <payload-size bytes>;
  reporting-rate seconds;
  transport protocol-name;
}
```

Hierarchy Level [edit services analytics]

Release Information Statement introduced in Junos OS Release 15.1F3.
payload-size *bytes* option introduced in Junos OS Release 16.1R3 on PTX Series routers only.

Description Configure the parameters of the export process for data generated through Junos Telemetry Interface sensors. You can create one or more export profiles. Each profile can be associated with one or more sensors that define the system resource to monitor and stream data. You can associate only one export profile with a specific sensor configuration.

The IP layer delivers the exported data to the remote server. The export profile configuration allows you to specify a format for exported data, a transport protocol, the rate which the system generates data, and the local source port and IP address that are used to define the transport headers in the exported packets.

To enable Junos Telemetry Interface, you must also configure a sensor that defines the parameters of the system resource to monitor and stream data, and a server to collect the data. To configure a sensor, include the sensor ***sensor-name*** statement at the **[edit services analytics]** hierarchy level. To configure the server that functions as a data collector, include streaming-server ***server-name*** statement at the **[edit services analytics]** hierarchy level.

NOTE: Junos Telemetry Interface was introduced in Junos OS Release 15.1F3 on MX Series routers with interfaces configured on MPC1 through MPC6E and on PTX Series routers with interfaces configured on FPC3. Starting in Junos OS Release 15.1F5, Junos Telemetry Interface is also supported on MPC7E, MPC8E, and MPC9E on MX Series routers.

Starting with Junos OS Release 16.1R3, FPC1 and FPC2 on PTX Series routers are also supported.

Options ***name***—Name of export profile.

NOTE: To associate this export profile with a configured sensor, include the name you configure for the export-profile statement at the [edit services analytics sensor *sensor-name* export-name] hierarchy level.

dscp *value*—Specify the DSCP value for the exported packets.

Range: 0 through 63.

Default: 0

format gpb—Specify the format to define the structure of exported data.

gpb—Google protocol buffers format.

forwarding-class (assured-forwarding | best-effort | expedited-forwarding | network-control)—(Packet Forwarding Engine sensors only) Specify the forwarding class for exported packets.

Default: best-effort

local-address *ip-address*—Specify the source address of exported packets.

local-port *number*—Specify the source port for the exported packets.

payload-size *bytes* (Optional) (PTX Series routers only)—Specify the maximum size of exported packets.

NOTE:

The payload-size option is supported only on the following sensors:

- /junos/system/linecard/interface/
- /junos/system/linecard/interface/logical/usage/
- /junos/system/linecard/firewall/

Default: 5000 bytes.

Range: 2000 through 9192 bytes.

NOTE: Junos Telemetry Interface does not export packets larger than 9192 bytes.

reporting-rate ***seconds***—Specify the interval at which the Junos Telemetry Interface sensor generates data to export to the collector.

As the configured interval expires, the most recent sample collected by the sensor is gathered and forwarded to the server configured to collect data.

NOTE: For Packet Forwarding Engine sensors, the minimum reporting rate is 2 seconds.

Range: 1 through 3600 (1 hour)

transport ***protocol-name***—Specify the transport protocol to use to carry the telemetry data in the IP packets.

udp—User Datagram Protocol.

Required Privilege Level

interface—To view this statement in the configuration.
interface-control—To add this statement to the configuration.

Related Documentation

- sensor on page 27

sensor (Junos Telemetry Interface)

Syntax

```
sesnor sensor-name {
  export-name export-profile-name;
  resource resource-string;
  <resource-filter regular expression>;
  server-name [ streaming-server-names ];
}
```

Hierarchy Level [edit services analytics]

Release Information Statement introduced in Junos OS Release 15.1F3.
Support for MPC7E, MPC8E, and MPC9E on MX Series routers added in Junos OS Release 15.1F5.
Support for FPC1 and FPC2 on PTX Series routers added in Junos OS Release 16.1R3.

Description Configure a Junos Telemetry Interface sensor, which defines the parameters of a system resource to monitor and stream data. You can configure more than one sensor to stream data for the same system resource. For example, you might want to configure different parameters for exporting data for the same system resource. Additionally, you can use regular expressions to filter the data collected. Examples include filters for logical and physical interfaces and LSP messages. To apply different filters to the same system resource, you configure multiple sensors. For example, you can configure multiple logical interface sensors and apply a different interface filter to each one.

Options Each sensor configuration requires you to specify the following: sensor name, an export profile name, a resource identifier string that enables monitoring and streaming of data for the specified system resource, and a server name to collect data. A regular expression to filter data for the specified resource is optional.

sensor-name—Specify a name that defines the sensor configuration. For example, for a sensor configuration that monitors all LSP events, you might choose the name **lsp-mon-global**. For a sensor configuration that monitors events only for an LSP named A2B, you might choose the name **lsp-mon-A2B**.

export-name *export-profile-name*—Specify the name of an export profile that you configured at the **[edit services analytics** export-profile ***name***] hierarchy level to associate with the sensor. This export profile defines the parameters for exporting telemetry data, such as a format for exported data and the rate at which data is generated for export.

NOTE: You can apply only one export profile to each sensor configuration.

The only supported transport protocol when you configure a sensor through the CLI is UDP.

resource *resource-string*—Enable the system resource to monitor and stream data. Each string corresponds to a specific system resource. The format is a file path and must

be entered exactly. You can associate only one ***resource-string*** with a ***sensor-name***. Configure a separate sensor for each system resource you want to monitor. The resource string to enable LSP monitoring can be modified to specify a specific LSP.

NOTE: You can configure more than one sensor to monitor the same system resource. Configuring different sensors for the same system resource allows you configure different parameters for monitoring that resource.

Table 4 on page 29 lists each supported ***resource-identifier-string***, a description of the system resource monitored, and additional configuration information.

Table 4: resource statement Options

resource string	Description	Release Information
/junos/services/label-switched-path/usage/	Packet Forwarding Engine sensor for LSP statistics. Only ingress LSPs are supported. Bypass LSPs, including those configured as ingress LSPs, are not supported. On MX Series routers only, bidirectional LSPs for ultimate-hop popping (UHP) are also supported. NOTE: You can modify **/junos/services/label-switched-path/usage/** to specify a specific LSP. Add **__instance__/*lsp-name*** to the end of the resource string identifier. For example, to monitor and stream data for LSP statistics for an LSP named mirror-to-murano-1, enter the following: **/junos/services/label-switched-path/usage/__instance__/mirror-to-murano-1**. If you do not specify a specific LSP name, the system resource monitors and streams data for all LSPs. When you enable a sensor for LSP statistics only, you must also configure the sensor-based-stats statement at the **[edit protocols mpls]** hierarchy level. On MX Series routers must also operate in enhanced mode. if not enabled by default, configure either the **enhanced-ip** statement or the **enhanced-ethernet** statement at the **[edit chassis network-services]** hierarchy level.	Junos OS Release 15.1F6 and later.
/junos/system/linecard/cpu/memory/	Packet Forwarding Engine sensor for CPU memory.	Junos OS Release 16.1R3 and later.
/junos/system/linecard/firewall/	Packet Forwarding Engine sensor for firewall filter counters and policer counters. Each line card reports counters separately. NOTE: Hierarchical policer statistics are collected for MX Series routers only. Traffic-class counter statistics are collected for PTX Series routers only. Firewall counters are exported even if the interface to which the firewall filer is attached is down.	Junos OS Release 15.1F5 and later.
/junos/system/linecard/interface/	Packet Forwarding Engine sensor for physical interface traffic. NOTE: For PTX Series routers, for a specific interface, queue statistics are exported for each line card. For MX series routers, interface queue statistics are exported only from slot on which an interface is configured. For Aggregated Ethernet interfaces, statistics are exported for the member physical interfaces. You must aggregate the counters at the destination server, or collector. If a physical interface is administratively down or operationally down, interface counters are not exported.	Junos OS Release 15.1F3 and later on PTX Series routers only. Support introduced for MX Series routers in Junos OS Release 15.1F5.

Table 4: resource statement Options *(continued)*

resource string	Description	Release Information
/junos/system/linecard/interface/logical/usage/	Packet Forwarding Engine sensor for logical interface traffic. NOTE: If a logical interface is operationally down, interface statistics continue to be exported.	Junos OS Release 15.1F5 and later.
/junos/system/linecard/npu/memory/	Packet Forwarding Engine sensor for network processing unit (NPU) memory.	Junos OS Release 16.1R3 and later.
/junos/system/linecard/npu/utilization/	Packet Forwarding Engine sensor for NPU memory utilization and total memory available for each memory type.	Junos OS Release 16.1R3 and later.
/junos/system/linecard/services/inline-jflow/	Packet Forwarding Engine sensor for performance metrics of the inline flow sampling process, such as the number of active flows and the number of exported flows.	Junos OS Release 16.1R3 and later.
/junos/system/linecard/optics/	Packet Forwarding Engine sensor for various optical performance metrics, such as transmit and receive power levels.	junos OS Release 16.1R3 and later.
/components/	Sensor for operational state of Routing Engines, power supply modules, Switch Fabric Boards, Control Boards, Switch Interface Boards, Modular Interface Cards, and Physical Interface Cards.	Junos OS Release 16.1R3 and later.

resource-filter ***regular-expression***—(Optional) Specify a regular expression to filter data for a specific resource. For example, you can filter for a specific set of logical or physical interfaces, firewall filters, or LSP messages. When you configure a system resource to monitor and stream data globally—that is, systemwide—you do not need to include a regular expression.

Examples of regular expressions to filter data exported through sensor configuration:

- Logical interface statistics sensor—et-2/0/7:1*
- LSP events sensor—lsp-from-A-to-B*
- Firewall filter counters sensor—f_testl*

server-name [***streaming- server-names*** **]** —Specify one or more servers to transport data for collection. Include at least one server-name configured at the **[edit services analytics** streaming-server ***server-name***] hierarchy level.

NOTE: Starting in Junos OS Release 15.1F6, you can configure as many as four streaming servers for a single sensor configuration. In previous releases, you can specify only one streaming server for each configured sensor. To specify more than one streaming server for a sensor, you must enclose the names in brackets.

Required Privilege Level

interface—To view this statement in the configuration.
interface-control—To add this statement to the configuration.

Related Documentation

- export-profile on page 24

sensor-based-stats (Junos Telemetry Interface)

Syntax sensor-based-stats;

Hierarchy Level [edit protocols mpls]

Release Information Statement introduced in Junos OS Release 15.1F6.

Description Enable the collection of LSP statstics for Junos Telemetry Interface. You must configure this statement when you configure a sensor to monitor and stream data for LSP statistics at the **[edit services analytics** sensor ***sensor-name*****]** hierarchy level. To enable the LSP statistics sensor, include the **resource /junos/services/label-switched-path/usage/** statement at the **[edit services analytics sensor *sensor-name*]** hierarchy level.

For additional information on configuring the LSP statistics sensor for Junos Telemetry Interface, see *Overview of LSP Statistics Export Format for Junos Telemetry Interface* and *Understanding How to Enable System Resource Monitoring*.

NOTE: This statement is supported only on MPCs on MX Series routers and on PTX Series routers.

Options None

Required Privilege Level routing—To view this statement in the configuration.
routing-control—To add this statement to the configuration.

Related Documentation

- Understanding the Junos Telemetry Interface Export Format of Collected Data on page 10

streaming-server (Junos Telemetry Interface)

Syntax

```
streaming-server streaming-server-name {
  remote-address ip-address;
  remote-port number;
}
```

Hierarchy Level [edit services analytics]

Release Information Statement introduced in Junos OS Release 15.1F3.

Description For Junos Telemetry Interface, configure the parameters of the server that collects exported data streamed by a monitored system resource. You can configure more than one streaming server. To collect data, you must associate a configured server with one or more configured sensors. The sensor configuration defines the parameters to monitor a specific system resource. To configure a sensor, include the sensor ***sensor-name*** statement at the **[edit services analytics]** hierarchy level.

To configure the server that collects data, you must also configure a destination IP address and a destination port. Junos Telemetry Interface relies on neighbor reachability information to deliver packets to the destination address. That means that all policies, such as filtering, that apply to the packets for that destination also apply to the exported packets.

NOTE: Starting with Junos OS Release 15.1F6. you can also associate more than one server with a specific sensor configuration, which enables you to transmit streamed data for the same sensor to more than one server.

NOTE: Junos Telemetry Interface was introduced in Junos OS Release 15.1F3 on MX Series routers with interfaces configured on MPC1 through MPC6E and on PTX Series routers with interfaces configured on FPC3. Starting in Junos OS Release 15.1F, Junos Telemetry Interface is also supported on MPC7E, MPC8E, and MPC9E on MX Series routers.

Starting with Junos OS Release 16.1R3, FPC1 and FPC2 on PTX Series routers are also supported.

Options ***streaming-server-name***—Specify a name for the server configured to collect data streamed through Junos Telemetry Interface. You can configure multiple streaming servers. To associate as many as four server names with a sensor configuration, include each name at the **[edit services analytics sensor *sensor-name* streaming server [*streaming-server-names*]]** hierarchy level. If you specify more than one streaming server, you must enclose the names in brackets.

remote-address ***ip-address***—Specify the destination address of the streaming server for exported packets.

remote-port ***number***—Specify a port number for the destination address of the streaming server for exported packets.

Required Privilege Level

interface—To view this statement in the configuration.
interface-control—To add this statement to the configuration.

Related Documentation

- export-profile (Junos Telemetry Interface) on page 24

show agent sensors

Syntax show agent sensors

Release Information Statement released in Junos OS Release 15.1F3

Description Display information about sensors configured for Junos Telemetry Interface.

NOTE: Junos Telemetry Interface was introduced in Junos OS Release 15.1F3 on MX Series routers with interfaces configured on MPC1 through MPC6E and on PTX Series routers with interfaces configured on FPC3. Starting in Junos OS Release 15.1F6, Junos Telemetry Interface is also supported on MPC7E, MPC8E, and MPC9E on MX Series routers.

Starting with Junos OS Release 16.1R3, FPC1, FPC2, and dual Routing Engines on PTX Series routers are also supported.

No other hardware is supported.

Required Privilege Level view

Related Documentation

- export-profile on page 24
- sensor on page 27
- streaming-server on page 33
- *Mapping OpenConfig Telemetry Commands to Junos OS Configuration*

List of Sample Output

show agent sensors (physical interface sensor) on page 36
show agent sensors (firewall filter sensor) on page 37

Output Fields Table 5 on page 35 lists the output fields for the **show agent sensors** command. Output fields are listed in the approximate order in which they appear.

Table 5: show agent sensors Output Fields

Field Name	Field Description
Sensor Information	Information about sensors configured to monitor system resources and stream data.
Name	Name of configured sensor.
Resource	Resource string used to configure and identify the system resource enabled to monitor and stream data..
Sensor-id	Numerical identifier of the sensor.
Server Information	Information about servers configured to collect sensor data.

Table 5: show agent sensors Output Fields *(continued)*

Field Name	Field Description
Name	Name of server.
Scope-id	
Remote-Address	Destination IP address for exported packets.
Remote-port	Destination port for exported packets.
Profile information	Information about export profiles for sensors.
Name	Name of export profile.
Rep-interval	Interval, in seconds, at which the sensor generates data to export.
Address	Source address of exported packets.
Port	Source port of exported packets.
Format	Format of exported data message: **GPB**
DSCP	Configured DSCP value for exported packets.
Forwarding-class	Configured forwarding class for exported packets.

Sample Output

show agent sensors (physical interface sensor)

```
user@host> show agent sensors
Sensor Information :
    Name                                : S1
    Resource                            : /junos/system/linecard/interface/

    Sensor-id                           : 2641

    Server Information :

        Name                            : JV5
        Scope-id                        : 0
        Remote-Address                  : 180.1.1.1
        Remote-port                     : 3008

    Profile Information :

        Name                            : EXP5
        Rep-interval                    : 2
        Address                         : 180.1.1.2
        Port                            : 30010
        Timestamp                       : 1
        Format                          : GPB
        Transport                       : UDP
```

```
        DSCP                              : 255
        Forwarding-class                  : 255
```

show agent sensors (firewall filter sensor)

```
user@host> show agent sensors
Sensor Information :

  Name                                    :firewall-stats
  Resource                                :/junos/system/linecard/firewall/
  Sensor ID                               :93390914

Server Information :

   Name                                   :jvision-server
   Scope ID                               :0
   Remote-Address                         :160.1.1.1
   Remote-port                            :2001

Profile Information :

  Name                                    :export-common
  Rep-interval                            :2
  Address                                 :160.1.1.2
  Port                                    :1000
  Timestamp                               :1
  Format                                  :GPB
  Transport                               :UDP
  DSCP                                    :255
  Forwarding-class                        :255
```

CHAPTER 5

Decdoing Data

Decoding Junos Telemetry Interface Data With UNIX Utilities

You can use UNIX utilities to decode Junos Telemetry Interface data on a server, or collector, that is streaming data from a Juniper Networks device. The example in this section shows you how to decode a single packet of streamed data.

Preparing the Collector to Decode Data

This example requires the following:

- UNIX OS with the Netcat (nc) utility.
- Protocol buffers compiler.
- Junos Telemetry Interface protocol buffers files.

This procedure shows how to prepare the collector to decode data using the Ubuntu OS.

1. Install the Netcat utility.

   ```
   sudo apt-get install netcat
   ```

2. Install the protocol buffers compiler.

   ```
   sudo apt-get install protobuf-compiler
   ```

3. Install the protocol buffers developer's library.

   ```
   sudo apt-get install libprotobuf-dev
   ```

4. Verify that the library files are installed.

   ```
   ls /usr/include/google/protobuf/descriptor.proto
   /usr/include/google/protobuf/descriptor.proto
   ```

5. Download and install the latest version of the Junos Telemetry interface protocol buffers files.

   From a Web browser, navigate to the All Junos Platforms software download URL on the Juniper Networks page: http://www.juniper.net/support/downloads/. After you select the name of the Junos OS platform and the release number, go to the **Tools** section and download the **Junos Telemetry Interface Data Model Files** package.

```
tar –xvzf junos-telemetry-interface-15.1F6.9.tgz
junos-telemetry-interface/telemetry_top.proto
junos-telemetry-interface/logical_port.proto
junos-telemetry-interface/lsp_mon.proto
junos-telemetry-interface/firewall.proto
junos-telemetry-interface/lsp_stats.proto
junos-telemetry-interface/port.proto
junos-telemetry-interface/NOTICE
junos-telemetry-interface/license.txt
```

NOTE: Be sure to note the location of the extracted files.

Decoding Data on the Collector

This procedure shows you how to capture data, decode raw data, and use the protocol buffers files to decode data.

To decode data:

1. Capture the data.

   Run netcat on a destination streaming telemetry server, or collector, in UDP listener mode to store all incoming datagrams into a file. Use the destination port number configured in streaming-server profile on your Juniper Networks device.

   ```
   nc -ul 0.0.0.0 20000 > data.gpb
   ```

   NOTE: This command stores datagrams into a file named **data.gpb**. Run this program to capture data. When you want to stop receiving data, stop with the program by sending the break signal (**Control + C**)

2. Decode raw data.

   NOTE: This step is optional. It is not required if you know the encoded message type of the data.

   Decode the message from the **data.gpb** file.

   ```
   protoc --decode_raw  < ../dump.gpb
   1: "hillrock:160.1.1.25"
   2: 0
   4:
   "S1:/junos/system/linecard/interface/logical/usage/:/junos/system/linecard/interface/logical/usage/:PFE"
   5: 65265
   6: 1477686534474
   7: 1
   8: 1
   101 {
     2636 {
       7 {
         1 {
   ```

```
        1: "et-0/0/4:2.32767"
        2: 1477642750
        3: 813
        4 {
          12: 0x37363732332e3165
        }
```

```
.
.
.
```

The next nested structure under **2636** identifies the sensor type. The numerical value **2636** identifies the **JuniperNetworksSensor** message, which is defined in the **telemetry_top.proto** file. In this example, the numerical identifier **7** corresponds to the **LogicalPort** message defined in the **logical_port.proto** file. Use this information in the next step to generate more detailed output.

3. Decode the message to include field names.

   Run the protocol buffers compiler with the decode option. Additionally, specify the top-level message type (**TelemetryStream**) and the file with the message definition, **logical_port.proto**. You must also include the Goggle protocol buffers (gpb) library.

```
protoc --decode TelemetryStream logical_port.proto -I /usr/include -I .  <
dump.gpb
system_id: "hillrock:160.1.1.25"
component_id: 0
sensor_name:
"S1:/junos/system/linecard/interface/logical/usage/:/junos/system/linecard/interface/logical/usage/:PFE"
sequence_number: 65268
timestamp: 1477686536484
version_major: 1
version_minor: 1
enterprise {
  [juniperNetworks] {
    [jnprLogicalInterfaceExt] {
      interface_info {
        if_name: "et-0/0/4:2.32767"
        init_time: 1477642750
        snmp_if_index: 813
        parent_ae_name: "ae1.32767"
        ingress_stats {
          if_packets: 0
          if_octets: 0
        }
        egress_stats {
          if_packets: 0
          if_octets: 0
        }
        op_state {
          operational_status: "up"
        }
      }
      interface_info {
        if_name: "et-0/0/7:3.0"
        init_time: 1477642750
        snmp_if_index: 520
        parent_ae_name: "ae0.0"
        ingress_stats {
          if_packets: 61203309
```

```
    if_octets: 6487548454
  }
  egress_stats {
    if_packets: 87416547
    if_octets: 9266153982
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.0"
  init_time: 1477642750
  snmp_if_index: 2512
  ingress_stats {
    if_packets: 26266247
    if_octets: 2784214806
  }
  egress_stats {
    if_packets: 26247215
    if_octets: 2781829290
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.1"
  init_time: 1477642750
  snmp_if_index: 2522
  ingress_stats {
    if_packets: 26266249
    if_octets: 2784214972
  }
  egress_stats {
    if_packets: 26249115
    if_octets: 2781935590
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.2"
  init_time: 1477642750
  snmp_if_index: 2523
  ingress_stats {
    if_packets: 26266248
    if_octets: 2784214912
  }
  egress_stats {
    if_packets: 26249106
    if_octets: 2781935086
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.3"
  init_time: 1477642750
```

```
  snmp_if_index: 2524
  ingress_stats {
    if_packets: 26266248
    if_octets: 2784214820
  }
  egress_stats {
    if_packets: 26248520
    if_octets: 2781902320
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.4"
  init_time: 1477642750
  snmp_if_index: 2525
  ingress_stats {
    if_packets: 26266247
    if_octets: 2784214760
  }
  egress_stats {
    if_packets: 26247302
    if_octets: 2781834112
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.5"
  init_time: 1477642750
  snmp_if_index: 2526
  ingress_stats {
    if_packets: 26266247
    if_octets: 2784214760
  }
  egress_stats {
    if_packets: 26247209
    if_octets: 2781828904
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.6"
  init_time: 1477642750
  snmp_if_index: 2527
  ingress_stats {
    if_packets: 26266248
    if_octets: 2784214820
  }
  egress_stats {
    if_packets: 26247196
    if_octets: 2781828226
  }
  op_state {
    operational_status: "up"
  }
}
```

```
interface_info {
  if_name: "et-0/0/13:0.7"
  init_time: 1477642750
  snmp_if_index: 2528
  ingress_stats {
    if_packets: 26266247
    if_octets: 2784214760
  }
  egress_stats {
    if_packets: 26247203
    if_octets: 2781828618
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.8"
  init_time: 1477642750
  snmp_if_index: 2529
  ingress_stats {
    if_packets: 26266247
    if_octets: 2784214760
  }
  egress_stats {
    if_packets: 26247225
    if_octets: 2781829850
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.9"
  init_time: 1477642750
  snmp_if_index: 2530
  ingress_stats {
    if_packets: 26266247
    if_octets: 2784214760
  }
  egress_stats {
    if_packets: 26247209
    if_octets: 2781828954
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.32767"
  init_time: 1477642750
  snmp_if_index: 648
  ingress_stats {
    if_packets: 4
    if_octets: 240
  }
  egress_stats {
    if_packets: 0
    if_octets: 0
  }
  op_state {
```

```
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/4:2.32767"
  init_time: 1477642750
  snmp_if_index: 813
  parent_ae_name: "ae1.32767"
  ingress_stats {
    if_packets: 0
    if_octets: 0
  }
  egress_stats {
    if_packets: 0
    if_octets: 0
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/7:3.0"
  init_time: 1477642750
  snmp_if_index: 520
  parent_ae_name: "ae0.0"
  ingress_stats {
    if_packets: 61206122
    if_octets: 6487846632
  }
  egress_stats {
    if_packets: 87420567
    if_octets: 9266580102
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.0"
  init_time: 1477642750
  snmp_if_index: 2512
  ingress_stats {
    if_packets: 26267458
    if_octets: 2784343172
  }
  egress_stats {
    if_packets: 26248420
    if_octets: 2781957020
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.1"
  init_time: 1477642750
  snmp_if_index: 2522
  ingress_stats {
    if_packets: 26267460
    if_octets: 2784343338
  }
```

```
  egress_stats {
    if_packets: 26250320
    if_octets: 2782063320
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.2"
  init_time: 1477642750
  snmp_if_index: 2523
  ingress_stats {
    if_packets: 26267459
    if_octets: 2784343278
  }
  egress_stats {
    if_packets: 26250311
    if_octets: 2782062816
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.3"
  init_time: 1477642750
  snmp_if_index: 2524
  ingress_stats {
    if_packets: 26267460
    if_octets: 2784343292
  }
  egress_stats {
    if_packets: 26249725
    if_octets: 2782030050
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.4"
  init_time: 1477642750
  snmp_if_index: 2525
  ingress_stats {
    if_packets: 26267459
    if_octets: 2784343232
  }
  egress_stats {
    if_packets: 26248507
    if_octets: 2781961842
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.5"
  init_time: 1477642750
  snmp_if_index: 2526
  ingress_stats {
```

```
    if_packets: 26267459
    if_octets: 2784343232
  }
  egress_stats {
    if_packets: 26248414
    if_octets: 2781956634
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.6"
  init_time: 1477642750
  snmp_if_index: 2527
  ingress_stats {
    if_packets: 26267460
    if_octets: 2784343292
  }
  egress_stats {
    if_packets: 26248401
    if_octets: 2781955956
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.7"
  init_time: 1477642750
  snmp_if_index: 2528
  ingress_stats {
    if_packets: 26267459
    if_octets: 2784343232
  }
  egress_stats {
    if_packets: 26248408
    if_octets: 2781956348
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.8"
  init_time: 1477642750
  snmp_if_index: 2529
  ingress_stats {
    if_packets: 26267459
    if_octets: 2784343232
  }
  egress_stats {
    if_packets: 26248430
    if_octets: 2781957580
  }
  op_state {
    operational_status: "up"
  }
}
interface_info {
  if_name: "et-0/0/13:0.9"
```

```
        init_time: 1477642750
        snmp_if_index: 2530
        ingress_stats {
          if_packets: 26267459
          if_octets: 2784343232
        }
        egress_stats {
          if_packets: 26248414
          if_octets: 2781956684
        }
        op_state {
          operational_status: "up"
        }
      }
      interface_info {
        if_name: "et-0/0/13:0.32767"
        init_time: 1477642750
        snmp_if_index: 648
        ingress_stats {
          if_packets: 4
          if_octets: 240
        }
        egress_stats {
          if_packets: 0
          if_octets: 0
        }
        op_state {
          operational_status: "up"
        }
      }
    }
  }
}
```

Related Documentation

- Configuring a Junos Telemetry Interface Sensor (CLI Procedure) on page 15

PART 3

Configuring gRPC Sensors

CHAPTER 6

OpenConfig and gRPC for Junos Telemetry Interface

Understanding OpenConfig and gRPC on Junos Telemetry Interface

Starting in Junos OS Release 16.1R3, you can use a set of remote procedure call (RPC) interfaces to configure the Junos Telemetry Interface and stream telemetry data using the gRPC framework. OpenConfig supports the use of vendor-neutral data models for configuring and managing multivendor networks. gRPC is an open source framework that provides secure and reliable transport of data.

NOTE: **OpenConfig for Junos OS and gRPC is supported only on MPCs on MX Series and on PTX Series routers.**

Network Agent Software

Implementing OpenConfig with gRPC for Junos Telemetry Interface requires that you download and install a package called Network Agent if your Juniper Networks device is running a version of Junos OS with Upgraded FreeBSD. For all other versions of Junos OS, the Network Agent functionality is embedded in the software. Network Agent functions as a gRPC server and terminates the OpenConfig RPC interfaces. It is also responsible for streaming the telemetry data according to the OpenConfig specification. To view the OpenConfig specification for telemetry, see the OpenConfig Telemetry specification. For more information about OpenConfig for Junos OS, see the *OpenConfig Feature Guide*.

The Network Agent component also supports server-based Secure Sockets Layer (SSL) authentication. Client-based SSL authentication is not supported. You must install SSL certificates on your Juniper Networks device.

For information about installing the Network Agent package, see "Installing the Network Agent Package" on page 54.

Using OpenConfig for Junos OS to Enable Junos Telemetry Interface

OpenConfig for Junos OS specifies an RPC model to enable the Junos Telemetry Interface. You must download and install the OpenConfig for Junos OS package on your Juniper Networks device. For more information see *Installing the Openconfig Module*. The programmatic interface **OpenConfigTelemetry** that is installed by the Network Agent package defines the telemetry gRPC service.

The **telemetrySubscribe** RPC specifies the following subscription parameters:

- OpenConfig path that identifies the system resource to stream telemetry data, for example:
  /interfaces/interface/state/counters/
- Interval at which data is reported and streamed to the collector server, in milliseconds, for example:
  sample_frequency = 4000

The **telemetrySubscribe** RPC is used by a streaming server, or collector, to request an inline subscription from for data at the specified path. The device should then send telemetry data back on the same connection as the subscription request.

For an example of how to use gRPC to subscribe to telemetry data, see "Example: gRPC Subscription for Telemetry Data" on page 61.

Using gRPC to Stream Data

Per the OpenConfig specification, only gRPC-based transport is supported for streaming data. The gRPC server that is installed by the Network Agent package terminates the gRPC sessions from the management system that runs the client. RPC calls trigger the creation of Junos OS sensors that either stream data periodically or report events, which are then funneled onto the appropriate gRPC channel by Network Agent.

See Table 6 on page 52 for a list and descriptions of the RPCs implemented to the support the Junos Telemetry Interface.

Table 6: Telemetry RPCs

RPC Name	Description
telemetrySubscribe	Specify telemetry parameters and stream data for the specified list of OpenConfig paths.
getTelemetrySubscriptions	Retrieve the list of subscriptions that are created through **telemetrySubscribe**.

Table 6: Telemetry RPCs *(continued)*

RPC Name	Description
cancelSubscription	Unsubscribe a subscription created through **telemetrySubscribe**.

To view the grRPC service definition, see the "gRPC Service Definition for OpenConfig Telemetry" on page 56.

Data streamed through gPRC is formatted in OpenConfig key/value pairs in Google protocol buffers (gpb) messages. In this universal format, keys are strings that correspond to the path of the system resources in the OpenConfig schema for the device being monitored. The values correspond to integers or strings that identify the operational state of the system resource, such a statistics counters and the state of a resource.

The following shows the universal key/value format:

```
message KeyValue {
    string key        = 1 [(telemetry_options).is_key = true];
    uint64 int_value  = 2;
    string str_value  = 3;
    string prefix_str = 4;
}

message TelemetryStream {
    // router name or export IP address
    required string system_id     = 1 [(telemetry_options).is_key = true];

    // line card / RE (slot number)
    optional uint32 component_id  = 2 [(telemetry_options).is_key = true];

   // PFE (if applicable)
    optional uint32 sub_component_id = 3 [(telemetry_options).is_key = true];

   // timestamp (common to all entries in the kv array)
    optional uint64 timestamp     = 4 [(telemetry_options).is_timestamp = true];


    // key / value pairs
    repeated KeyValue kv;
}
```

The following example shows how a set of counters for an interface can be represented:

```
key = "/interfaces/counters/rx-bytes",     int_value = 1000
key = "/interfaces/counters/tx-bytes",     int_value = 2000
key = "/interfaces/counters/rx-packets", int_value = 10
key = "/interfaces/counters/rx-bytes" ,    int_value = 20
key = "/interfaces/counters/oper-state", str_value = "up"
```

The Network Agent package provides a mapping table that maps field names to the OpenConfig key strings.

Release History Table

Release	Description
16.1R3	Starting in Junos OS Release 16.1R3, you can use a set of remote procedure call (RPC) interfaces to configure the Junos Telemetry Interface and stream telemetry data using the gRPC framework.

Related Documentation

- Installing the Network Agent Package (Junos Telemetry Interface) on page 54
- *Understanding Junos OS with Upgraded FreeBSD*

Installing the Network Agent Package (Junos Telemetry Interface)

Starting with Junos OS Release 16.1R3, the Network Agent software package provides a framework to support OpenConfig and gRPC for the Junos Telemetry Interface. The Network Agent package functions as a gRPC server that terminates the OpenConfig remote procedure call (RPC) interfaces and streams the telemetry data according to the OpenConfig specification. The Network Agent package, which runs on the Routing Engine, implements local statistics collection and reports data to active telemetry stream subscribers.

Network Agent is available as a separate package only for Junos OS with Upgraded FreeBSD. For other versions of Junos OS, Network Agent functionality is embedded in the software. For more information about Junos OS with Upgraded FreeBSD, see *Understanding Junos OS with Upgraded FreeBSD*.

Network Agent for Junos OS software package has the following naming conventions:

- Package Name—This is **Network-Agent**.
- Architecture—This field indicates the CPU architecture of the platforms, such as **x86**.
- Application Binary Interface (ABI)—This field indicates the "word length" of the CPU architecture. Vales include **32** for 32-bit architectures and **64** for 64-bit architectures.
- Release—This field indicates the Junos OS release number, such as **16.1R3.16**.
- Package release and spin number—This field indicates the package version and spin number, such as **C1.1**.

All Network Agent packages are in tarred and gzipped (**.tgz**) format.

NOTE: Each version of the Network Agent package is supported on a single release of Junos OS only. The Junos OS version supported is identified by the Junos OS release number included in the Network Agent package name.

Examples of valid Network Agent package names including the following:

- **network-agent-x86-64-16.1R3.16-C1.0.tgz**
- **network-agent-x86-32-16.1R4.12-C1.1.tgz**

Before you begin:

- Install Junos OS Release 16.1R3 or later.
- Install the OpenConfig for Junos OS module. For more information, see *Installing the Openconfig Module*.
- Install Secure Sockets Layer (SSL) certificates of authentication on your Juniper Networks device.

NOTE: Only server-based SSL authentication is supported. Client-based authentication is not supported.

To download and install the Network Agent package:

1. Using a Web browser, navigate to the All Junos Platforms software download URL on the Juniper Networks webpage: http://www.juniper.net/support/downloads/.
2. Select the name of the Junos OS platform for the software that you want to download.
3. Select the release number (the number of the software version that you want to download) from the **Release** drop-down list to the right of the Download Software page.
4. Select the **Software** tab.
5. In the **Tools** section of the **Software** tab, select the **Junos Network Agent** package for the release.
6. Log in to the Juniper Networks authentication system using the username (generally your e-mail address) and password supplied by a Juniper Networks representative.
7. Download the software to a local host.
8. Copy the software to Juniper Networks device or to your internal software distribution site.
9. Install the new **network-telemetry** package on the device by issuing the **request system software add *package-name*** from the operational mode:

   For example:

   ```
   user@host > request system software add
     network-telemetry-x86-64-16.1R3.16-C1.0.tgz
   ```

NOTE: The command uses the validate option by default. This option validates the software package against the current configuration as a prerequisite to adding the software package to ensure that the device reboots successfully. This is the default behavior when the software package being added is a different release.

Replace ***source*** with one of the following values:

- ***/pathname***—For a software package that is installed from a local directory on the device.
- For software packages that are downloaded and installed from a remote location:
  - ***ftp://hostname/pathname***
  - ***http://hostname/pathname***
  - ***scp://hostname/pathname*** (available only for Canada and U.S. version)

10. Issue the **show version** command to verify that the Network Agent package was successfully installed.

Related Documentation

- Understanding OpenConfig and gRPC on Junos Telemetry Interface on page 51

gRPC Service Definition for OpenConfig Telemetry

Like many RPC systems, gRPC is based is based on defining a service, specifying the methods that can be called remotely with their parameters and return types. By default, gRPC uses protocol buffers as the Interface Definition Language (IDL) for describing both the service interface and the structure of the payload messages.

Following is the gRPC service definition for the **telemetrySubscribe** RPC:

```
//
//  na-openconfig-telemetry.proto
//  network-agent-gprc
//     This file defines the telemetry gRPC service implemented by  the
//     Network Agent that runs on Juniper devices.
//
syntax = "proto3";
package Telemetry;

service OpenConfigTelemetry {
    // Request an inline subscription for data at the specified path
    rpc telemetrySubscribe(SubscriptionRequest) returns (SubscriptionResponse)
{}
    // Get the list of current telemetry subscriptions from the
    // target.  This command returns a list of existing subscriptions
    // not including those that are established via configuration."
    rpc getTelemetrySubscriptions() returns (SubscriptionResponse) {}
    // Terminates and removes an existing telemetry subscription
    rpc cancelTelemetrySubscription(SubscriptionId) returns () {}
    // Retrieve operational state values of the given OC paths
    rpc getOperational(OpDataRequest) returns (OpDataReply) {}
    // Get the capabilities of the network device
    rpc getCapabilities() returns (capabilities) {}
}

message SubscriptionRequest {
    // List of optional collector endpoints to send data for
    // this subscription, specified as an ip+port combination.
    // If no collector destinations are specified, the collector
    // destination is inferred from requester on the rpc channel
```

```
    repeated Collector collectors = 1;
    // The DSCP code point to be set on telemetry messages
    uint32   export_dscp_marking  = 2;
    // List of paths for which telemetry is desired
    repeated Resource resources    = 3;
}

message Collector {
    // IP address of collector end point
    string ip_address   = 1;
    // Transport protocol port number for destination
    uint32 port = 2;
}

message Resource {
    // Datamodel path of interest
    Path path               = 1;

    // Regular expression used to filter out non interesting leaf nodes
    string filter           = 2;

    // The interval at which the value of a counter is reported
    int32  sample-frequency = 3;
}

message SubsscriptionResponse {
    SubscriptionId      id                 = 1;
    SubscriptionRequest actualSubscription = 2;
}

message SubscriptionId {
    uint32 id = 1;
}

message OpDataRequest {
    // Request header
    RequestHeader header = 1;

    // List of paths
    repeated Path paths = 2;

    // True if only operational paths need to be retrieved
    unit32 opOnly = 3;
}

message RequestHeader {
    // Request ID
    uint64 reqId = 1;
}

message Path {
    // Name of the Path
    string path = 1;
}

message OpDataReply {
    // Reply header
    ReplyHeader header = 1;
    // Response in path-value format
    repeated OpData keyvalues = 2;
}
```

```
message ReplyHeader {
    // Request ID
    uint64 reqId = 1;
    // Response code, success or failure
    uint32 rspCode = 2;
    // Info or error message
    string rspMsg = 3;
}

message OpData {
    // Name of the path
    string path = 1;
    // Value of the path
    string value = 2;
}

message Capabilities {
    // List of the paths
    repeated Capability capabilities = 1;
}

message Capability {
    // The resource path
    Path               path = 1;
    // The recommended polling interval
    uint32 polling_interva; = 2;
    // Whether this path supports filtering with regular expressions
    string filter_supported = 3;
}
```

Related Documentation

- Understanding OpenConfig and gRPC on Junos Telemetry Interface on page 51

Guidelines for gRPC Sensors

Starting with Junos OS Release 16.1R3, the Junos Telemetry Interface supports gRPC remote procedure calls (gRPC) to provision sensors and to subscribe to and receive telemetry data. The following sensors are supported with gRPC. To activate a sensor, use the corresponding resource path. Each resource path enables data streaming for the system resource globally, that is, systemwide. You can also modify resource path, such as to specify a specific logical or physical interface. For example, to specify a specific interface, include the following at the end of the path: **[name='interface-name']/**

Supported gRPC Sensors

See Table 7 on page 59 for a description of supported gRPC sensors and the subscription path you use to provision the sensors.

Table 7: gRPC Sensors

resource path	Description
/junos/services/label-switched-path/usage/	Sensor for LSP statistics. Only ingress LSPs are supported. On MX Series routers only, the following are also supported: bypass LSPs, including those configured as ingress LSPs, and bidirectional LSPs for ultimate-hop popping (UHP). NOTE: You can modify **/junos/services/label-switched-path/usage/** to specify a specific LSP. Add **__instance__/*lsp-name*** to the end of the resource path. For example, to monitor and stream data for LSP statistics for an LSP named mirror-to-murano-1, enter the following: **/junos/services/label-switched-path/usage/ __instance__/mirror-to-murano-1**. If you do not specify a specific LSP name, the system resource monitors and streams data for all LSPs. When you enable a sensor for LSP statistics only, you must also configure the sensor-based-stats statement at the **[edit protocols mpls]** hierarchy level. MX Series routers should operate in enhanced mode. If not enabled by default, include either the **enhanced-ip** or the **enhanced-ethernet** statement at the **[edit chassis network-services]** hierarchy level.
/junos/npu-memory/	Sensor for network processing unit (NPU) memory, NPU memory utilization, and total memory available for each memory type
/junos/system/linecard/cpu/memory/	Sensor for CPU memory.
/bgp/neighbors/neighbor/	Sensor for BGP peer information. NOTE: You can also include the following at the end path: • **state/session-state/** • **state/messages/sent/update/** • **state/messages/received/update/** • **timers/state/uptime/** • **transport/state/local-address/** • **transport/state/remote-address/** • **state/peer-as/** • **afi-safis-afi/safi/state/prefix-limit/state/max-prefixes/** • **state/session-status** • **state/session-admin-status** • **state/session-established-transitions/** • **state/interface-error/** • **state/prefix-limited/exceeded/**
/junos/services/routing/task-memory-utilization/	Sensor for memory utilization for routing protocol task.

Table 7: gRPC Sensors *(continued)*

resource path	Description
/junos/system/linecard/firewall/	Sensor for firewall filter counters and policer counters. Each line card reports counters separately. NOTE: Hierarchical policer statistics are collected for MX Series routers only. Traffic-class counter statistics are collected for PTX Series routers only. Firewall counters are exported even if the interface to which the firewall filter is attached is operationally down.
/junos/system/linecard/interface/	Sensor for physical interface traffic. NOTE: For PTX Series routers, for a specific interface, queue statistics are exported for each line card. For MX series routers, interface queue statistics are exported only from slot on which an interface is configured. For Aggregated Ethernet interfaces, statistics are exported for the member physical interfaces. You must aggregate the counters at the destination server, or collector. If a physical interface is administratively down or operationally down, interface counters are not exported.
/interfaces/interface/subinterfaces/ **/interfaces/interface[name='*interface-name*']/subinterfaces/**	Sensor for logical interface traffic. NOTE: If a logical interface is operationally down, interface statistics continue to be exported.
/junos/system/linecard/optics/	Sensor for various optical interface performance metrics, such as transmit and receive power levels.
/junos/rsvp-interface-information/	Sensor for events and properties for RSVP interfaces. NOTE: For 100 RSVP logical interfaces, configure a sampling interval equal to 60 seconds. For 200 RSVP logical interfaces, configure a sampling interval equal to 180 seconds.
/components/	Sensor for operational state of Routing Engines, power supply modules, Switch Fabric Boards, Control Boards, Switch Interface Boards, Modular Interface Cards, and Physical Interface Cards.
/lacp/	Sensor for operational state of aggregated Ethernet interfaces configured with the Link Aggregation Control Protocol.
/lldp/	Sensor for operational state of Ethernet interfaces enabled with the Link Layer Discovery Protocol.
/arp-information/	Sensor for Address Resolution Protocol (ARP) statistics.

Table 7: gRPC Sensors *(continued)*

resource path	Description
/interfaces/interface[name='*interface-name*']/	Sensor for Routing Engine internal interfaces. NOTE: On MX Series routers, you can specify the following interfaces: fxp0, em0, and em1 On PTX Series routers, you can specify the following interfaces: em0, ixlv0, ixlv1 On PTX Series routers with dual Routing Engines, you can specify the following interfaces: em0, ixgbe0, ixgbe1
/nd6-information/	Sensor for Network Discovery Protocol (NDP) table state.
/ipv6-ra/	Sensor for NDP router-advertisement statistics.

Related Documentation

- Understanding OpenConfig and gRPC on Junos Telemetry Interface on page 51
- Example: gRPC Subscription for Telemetry Data on page 61

Example: gRPC Subscription for Telemetry Data

This example shows how to use gRPC remote procedure calls (gRPC) to provision a sensor for logical interface statistics and to subscribe to and receive telemetry data. A management station serves as the collector for telemetry data streamed from the Juniper Networks device.

Before you begin, complete the following steps on your Juniper Networks device:

- Download and install Junos OS Release 16.1R3 or later.
- Download and install the OpenConfig for Junos OS module.
- If your Juniper Networks device is running a version of the Junos OS with the upgraded FreeBSD kernel, download and install the Network Agent package.

In this example, the management station requesting telemetry data has Python programming language modules installed. Python is used to write a program to specify the following parameters of the gRPC subscription request:

- Path of the system resource—**/interfaces**. This path specifies to stream data for all logical interfaces.
- Interval at which to report data, in milliseconds—**sample_frequency = 4000**

The **telemetrySubscribe** RPC is used to initiate the subscription. Data is returned in an OpenConfig key/value format in a Google protocol buffers (gpb) message.

Telemetry Python Client Code

```
ip = "10.209.16.147"
port = 50051
_TIMEOUT_SECONDS = 55000000

def collectData (ip, port):
    channel = implementations.insecure_channel(ip, port)
    with agent_pb2.beta_create_OpenConfigTelemetry_stub(channel) as stub:
      try:
        # From subscription request for /interfaces with reporting rate
        # of 4000 millisecs
        # PS: It should be noted that SubscriptionRequest can take list of
        #     paths and their reporting rates. Here just for a simple example,
        #     only one path is demonstrated
        sub_req = agent_pb2.SubscriptionRequest()
        path_list = sub_req.path_list.add()
        path_list.path = "/interfaces"
        path_list.sample_frequency = 4000

        # Request: Subscribe for the above path and get the below resplies
        #       1. List of accespted paths and the reporting rate
        #       2. OpenConfig data
        data_itr = stub.telemetrySubscribe(sub_req, _TIMEOUT_SECONDS)

        # Reply 1: SubscriptionReply - Get the list of accepted paths as part
        #          of telemetrySubscribe. This will be filled as meta-data
        #          in the gRPC channel as a key-value pair with
        #          "init-response" as key

        metadata = data_itr.initial_metadata()
        if metadata:
              if metadata[0][0] == "init-response":
                metainfo = metadata[0][1]

                # Format the meta-data to SubscriptionReply
                subreply = agent_pb2.SubscriptionReply()
                subreply.SetInParent()
                google.protobuf.text_format.Merge(metainfo, subreply)
                print "\r\n"
                print "Subscription Reply in the form of Meta data"
                print "-------------------------------------------"
                print   subreply

        # Reply 2: OpenConfigData - Get the OpenConfig data
        #          streamed from the device
        print "\r\n"
        print "OpenConfigData reply as streaming response"
        print "------------------------------------------"
        for data in data_itr:
              if not data:
                print "Server returned NULL"
                next
              oc_data = ParseOpenConfigData(data)
              print oc_data

      except Exception, e:
        print 'Exception: ' + str(e)

def ParseOpenConfigData(data):
    msg = ""
```

```
    #if data.system_id :
    msg += "\nsystem_id:" + data.system_id + "\n"
    #if data.component_id:
    msg += "component_id:" + str(data.component_id) + "\n"

    if data.sub_component_id:
      msg += "sub_component_id:" + str(data.sub_component_id) + "\n"

    if data.path:
      msg += "path:" + data.path + "\n"

    msg += "sequence_number:" + str(data.sequence_number) + "\n"

    if data.timestamp:
     msg += "timestamp:" + str(data.timestamp) + "\n"

    for kv in data.kv:
      msg += "kv {\n"
      if kv.key :
        msg += "  " + "key:" + kv.key + "\n"

      oneofType = kv.WhichOneof("value")
      if oneofType == "int_value" :
        msg += "  " + "int_value:" + str(kv.int_value) + "\n"
      elif oneofType == "double_value" :
        msg += "  " + "double_value:" + str(kv.double_value) + "\n"
      elif oneofType == "uint_value" :
        msg += "  " + "uint_value:" + str(kv.uint_value) + "\n"
      elif oneofType == "sint_value" :
        msg += "  " + "sint_value:" + str(kv.sint_value) + "\n"
      elif oneofType == "bool_value" :
        msg += "  " + "bool_value:" + str(kv.bool_value) + "\n"
      elif oneofType == "str_value" :
        msg += "  " + "str_value:" + kv.str_value + "\n"
      elif oneofType == "bytes_value" :
        msg += "  " + "bytes_value:" + str(kv.bytes_value) + "\n"
      msg += "}\n"

    return msg

def run():
  collectData(ip, port)

if __name__ == '__main__':
  run()
```

After you initiate the subscription using this program, the following subscription reply is returned:

```
response {
  subscription_id: 1
}
path_list {
  path: "/interfaces"
  sample_frequency: 4000
}
```

Data is then streamed to the management station console as shown below:

```
system_id:choc-mx240-f
component_id:1
```

```
path:sensor_1000_1_2:/junos/system/linecard/interface/logical/usage/:/junos/system/linecard/interface/logical/usage/:PFE
sequence_number:0
timestamp:1477077921468
kv {
  key:__timestamp__
  uint_value:1477077921434
}
kv {
  key:__prefix__
  str_value:/interfaces/interface[name='ge-1/0/0']/subinterfaces/subinterface[index='0']/
}
kv {
  key:init-time
  int_value:1475784511
}
kv {
  key:operational-state
  str_value:down
}
kv {
  key:__prefix__
  str_value:/interfaces/interface[name='ge-1/0/0']/subinterfaces/subinterface[index='1']/
}
.
.
.
.
.
.
```

Related Documentation

- Understanding OpenConfig and gRPC on Junos Telemetry Interface on page 51

PART 4

Best Practices

CHAPTER 7

Best Practices for Implementing the Junos Telemetry Interface

Guidelines for Specifying Data Reporting Intervals Junos Telemetry Interface

The Junos Telemetry Interface enables you to provision sensors to collect and export data for various system resources without involving polling. A request to send data is sent once by a management station to stream periodic updates.

You can configure telemetry sensors to report data at a specified interval either through the command-line interface (CLI) or through the OpenConfig for Junos **telemetrySubscribe** remote procedure call (RPC). To configure using the CLI, include the **reporting-rate** *seconds* statement at the **[edit services analytics** export-profile *profile-name***]** hierarchy level. For the **telemetrySubscribe** RPC, specify the sampling interval parameter, in milliseconds. In both cases, the interval specifies the amount of time between each subsequent export of data.

How to Determine the Reporting Interval for a System Resource

To determine the appropriate reporting interval for a specific system resource, follow these guidelines:

- Identify the required export interval for a given object, such as an interface.
- Identify the maximum number of objects reported by the sensor, such as the number of physical interfaces configured on a line card.
- Identify the minimum number of objects reported on each interval for a given sensor.
- Use the following formula to determine the best reporting interval:
  - Reporting interval = Required Export Interval Per Object * Minimum Number of objects reported on each Interval / Maximum Number of Objects.

Consider this example. There is a business requirement to report interface statistics every 30 seconds. At every interval, 10 interface records are reported, and the total number of interfaces is 96 for each line card. Using the reporting-interval formula, the reporting

interval should be 3.125 seconds. Currently, the reporting interval can be configured only as a multiple of 2, in seconds. Therefore, for this example, configure the reporting interval as 2 seconds in the CLI or 2000 milliseconds in the OpenConfig RPC.

TIP: The same metric might be reported more than once over a 30-second interval. For the purposes of effective visualization and data manipulation, it is quite common to aggregate data over fixed time spans.

Related Documentation

- Overview of the Junos Telemetry Interface on page 3

Guidelines for Aggregating Junos Telemetry Interface Data

One important feature of the Junos Telemetry Interface is that data processing occurs at the collector that streams data, rather than the device. Data is not automatically aggregated, but it can be aggregated for analysis.

Data aggregation is useful in the following scenarios:

- Data for the same metric over fixed spans of time, such as, the average number physical interface ingress errors over a 30-second interval.
- Data from different sources (such as multiple line cards) for the same metric, such as label-switched path (LSP) statistics or filter counter statistics.
- Data from multiple sources, such as input and output statistics for aggregated Ethernet interfaces.

The follow sections describe how to perform data aggregation for various scenarios. The examples in these sections use the InfluxDB time-series database to accept queries on telemetry data. InfluxDB is an open source database written in Go specifically to handle time-series data.

Aggregating Data Over Fixed Time Spans

Aggregating data for the same metric over fixed spans of time is a common and useful way to detect trends. Metrics can include gauges, that is, single values, or cumulative counters. You might also want to aggregate data continuously.

Example: Aggregating Data for Gauge Metrics

In this example, data for **JuniperNetworksSensors.jnpr_interface_ext.interface_stats.egress_queue_info.current_buffer_occupancy** from **port.proto** is written to the InfluxDB database with tags that identify the host name, an interface name and corresponding queue number and measurement called **current_buffer_occupancy**. See Table 8 on page 69 for the specific values used in this example.

Table 8: Telemetry Data Values

Time Stamp (seconds)	Value	Tags
1458704133	1547	queue_number=0,interface_name='xe-1/0/0',host='sjc-a'
1458704143	3221	queue_number=0,interface_name='xe-1/0/0',host='sjc-a'
1458704155	4860	queue_number=0,interface_name='xe-1/0/0',host='sjc-a'
1458704166	6550	queue_number=0,interface_name='xe-1/0/0',host='sjc-a'

Each measurement data point has a timestamp and recorded value. In this example, the tag **queue_number** is the numerical identifier of the interface queue.

To aggregate this data over 30-second intervals, use the following influxDB query:

```
select mean(value) from current_buffer_occupancy
       where  time >= $time_start and time <= $time_end and
              queue_number='0' and interface_name='xe-1/0/0' and host='sjc-a'
       group by time(30s)
```

For **$time_start** and **$time_end**, specify the actual range of time.

Example: Aggregating Data for Cumulative Statistics

Some Junos Telemetry Interface sensors report cumulative counter values, such as the number of ingress packets, defined as **JuniperNetworksSensors.jnpr_interface_ext.interface_stats.ingress_stats.packets**.

It is common to derive traffic rates from packet or byte counters. Unlike with gauge metrics, the initial data point in the series for cumulative counters is used only to set the baseline.

Use the following guidelines to create a database query for cumulative statistics:

- Calculate the cumulative value for a specific time interval. You can calculate either an average among several data points recorded during the time interval, or you can interpolate a value. All data points should belong to the same series. If a counter reset has occurred between the two data points reported at different times, do not use both data points.
- Determine the appropriate value for the previous time interval. If a counter has been reset since the last update, declare that value as unavailable.
- If the previous interval is available, calculate the difference between the data points and the traffic rate.

These guidelines are summarized in the following influxDB query. This query assumes that data is stored in the measurement **ingress_packets**. The query uses the same tags as the gauge metric example as well as the tag for counter initialization time, **init_time**.

The query uses average values over a 30-second time interval. It calculates the rate for the metrics that have the same counter initialization.

```
select non_negative_derivative(mean(value)) from ingress_packets
        where  time >= $time_start and time <= $time_end and
               interface_name='xe-1/0/0' and host='sjc-a'
        group by time(30s), init_time
```

Use the following query to calculate the number of packets received over an interval of time, without deriving the rate.

```
select difference(mean(value)) from ingress_packets
        where  time >= $time_start and time <= $time_end and
               interface_name='xe-1/0/0' and host='sjc-a'
        group by time(30s), init_time
```

In some cases, more than one aggregated data point is returned by the query for a particular time interval. For example, four data points are available for a time interval. Two data points have **init_time t0**, and the other two have **init_time t1**. You can run a query that uses the last change timestamp tag, **last_change**, instead of **init_time**, to calculate the difference and to derive the rate between the two data points with the same last change timestamp.

```
select difference(mean(value)) from ingress_packets
       where  time >= $time_start and time <= $time_end and
              interface_name='xe-1/0/0' and host='sjc-a'
       group by time(30s), last_change
```

TIP: These queries can all be run as continuous queries and can periodically populate new time-series measurements.

Aggregating Data From Multiple Sources

Certain metrics are reported from multiple line cards or packet forwarding engines. It is useful to aggregate data derived from different sources in the following scenarios:

- Packet and byte counts for label-switched paths (LSPs) are reported separately by each line card. However, a view of LSP paths for the entire device is required for path computation element controllers.
- For Juniper Networks devices that support virtual output queues, the tail drop or random early detection drop statistics for each queue are reported separately by each line card for every physical interface. It is useful to be able to aggregate the statistics for all the line cards for an interface.
- Filter counters for a firewall filter attached to a forwarding table or to an aggregated Ethernet interface are reported separately by each line card. It is useful to aggregate the statistics for all the line cards.

To aggregate data from multiple sources, perform the following:

1. Aggregate data for a specific period of time for each source, for example, each line card.

2. Aggregate the data you derive for each source in *step 1*.

For data stored in an InfluxDB database, you can complete *step 1* in the procedure by running a continuous query and populating a new measurement. We strongly recommend that you group the data points according to each source. For example, for LSP statistics, the **component_id** in the the gpb message identifies the line card sending the data. Group the data points based on each unique **component_id**.

Example: Aggregating Data from Multiple Sources

In this example, you run two queries to derive the LSP packet rate for data from all line cards.

First, you run the following continuous query on the measurement named **lsp_packet_count** for each **component_id** tag and the **counter_name** tag. Each unique **component_id** tag corresponds to a different line card. This query populates a new measurement, **lsp_packet_rate.**

```
select non_negative_derivative(mean(value)) as value from lsp_packet_count
       into lsp_packet_rate
       group by time(30s), component_id, counter_name, host
```

NOTE: The LSP statistics sensor does not report counter initialization time.

Use the new measurement derived from this continuous query—**lsp_packet_count**—to run the following query, which aggregates data from all line cards for packet rates for an LSP named **lsp-sjc-den-1**.

```
select sum(value) from lsp_packet_rate
       where counter_name='lsp-sjc-den-1', host='sjc-a'
```

NOTE: Because this query does not group data according to the component_id tag, or line card, the LSP packet rates from all components, or line cards, are returned.

Aggregating Data for Multiple Metrics

It can be useful to aggregate metrics for multiple values. For example, for aggregated Ethernet interfaces, you would typically want to track packet and byte rates for each interface member as well as interface utilization for the aggregated link.

Example: Aggregating Multiple Metric Values

In this example, you run the following two queries:

- Continuous query to derive ingress packet counts for each member link in an aggregated Ethernet interface
- Query to aggregate packet count data for all the member links that belong to the same aggregated Ethernet interface

The following continuous query derives a measurement, **ingress_packets**, for each member link in an aggregated Ethernet interface. The **interface_name** tag identifies each member interface. You also use the **parent_ae_name** tag to identify membership in a specific aggregated Ethernet interface. Grouping each member link with the **parent_ae_name** tag ensures that data is collected only for current member links. For example, an interface might change its membership during the reporting interval. Grouping member interfaces with the specific aggregated Ethernet interface means that data for the member link will not be transferred to the new aggregated Ethernet interface of which it is now a member.

```
select difference(mean(value)) as value from ingress_packets
       into ingress_packets_difference
       group by time(30s), component_id, interface_name, host, parent_ae_name
```

The following query aggregates data for the ingress packets for the aggregated Ethernet interface, that is all member links.

```
select sum(value) from ingress_packets_difference
       where parent_ae_name='ae0' and host='sjc-a'
```

NOTE: This query aggregates data for aggregated Ethernet interface ae0. The parent_ae_name tag does not verify the actual member links.

Related Documentation

- Overview of the Junos Telemetry Interface on page 3