

Junos OS

Junos Telemetry Interface User Guide

Published
2025-06-22

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

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Junos OS Junos Telemetry Interface User Guide

Copyright © 2025 Juniper Networks, Inc. All rights reserved.

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

YEAR 2000 NOTICE

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

END USER LICENSE AGREEMENT

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

Table of Contents

About This Guide | viii

1

Junos Telemetry Interface

Understanding Junos Telemetry Interface | 2

2

Dial-in Telemetry

Understanding Dial-in Telemetry | 21

gRPC Service | 22

gRPC Server Overview | 25

gNMI Service | 26

Guidelines for Telemetry Data Subscriptions over gNMI | 29

Examples | 35

Configure gRPC Services | 51

Understanding Authentication and Authorization for gRPC-Based Services | 52

Obtain X.509 Certificates | 54

Load the gRPC Server's Local Certificate in the Junos PKI | 57

Enable gRPC Services | 58

Configure Mutual (Bidirectional) Authentication for gRPC Services | 63

Configure the User Account for gRPC Services | 68

Configure gRPC RPC Authorization | 69

Establish a Dial-in Telemetry Connection | 71

Configure the Data Collector | 72

Selecting Sensor Paths | 73

Decoding Data on the Collector | 74

3

Dial-out Telemetry

Understanding Dial-Out Telemetry | 88

Guidelines for Streaming Telemetry Data Over UDP | 89

Configure gRPC service for Legacy Dial-Out Connection | 91

Configure an IP Source Address for Legacy gRPC Dial-Out Connections | 91

Configure a Routing Instance for Legacy gRPC Dial-Out Connections | 92

Establish a Dial-out Telemetry Connection | 94

Configure a Streaming Server Profile | 95

Configure a Sensor Profile | 96

Configure an Export Profile | 98

Configure the Data Collector | 103

Verify the Junos Telemetry Interface Sensor Configuration | 103

Selecting Sensor Paths | 105

Decoding Junos Telemetry Interface Data With UNIX Utilities | 106

Junos Telemetry Interface Features

gNMI Genstate Subscription | 124

gRPC Tunnels Overview | 126

Example: Configure a gRPC Tunnel | 128

Overview | 128

Requirements | 128

Topology | 128

Configure a gRPC tunnel | 129

End-of-Message Notification for Routing Engine Sensor | 133

Enabling Client Streaming and Bidirectional Streaming of Telemetry Sensor Information | 134

Configure a NETCONF Proxy Telemetry Sensor in Junos | 135

Create a User-Defined YANG File | 139

Load the Yang File in Junos | 144

Collect Sensor Data | 145

Installing a User-Defined YANG File | 148

Troubleshoot Telemetry Sensors | 149

Support for Zero Suppression | 151

Configure SR-TE for Uncolored Ingress LSPs | 151

Supporting Uncolored Sensor Paths | 152

Configure SR-TE for Uncolored SR-TE Tunnel Statistics | 153

Configure SR-TE for Uncolored SR-TE Per-tunnel Statistics | 153

Per-Path or Per-Segment-List Traffic Statistics | 154

Configuring Per-Path Traffic Statistics | 155

SRv6 and SRv6-TE Traffic Sensor Telemetry | 159

On-Box Aggregation Overview | 163

Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets | 169

Understanding Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets | 169

Enable Export of Subscriber Statistics and Queue Statistics | 171

Guidelines for Exporting Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets | 173

gRPC Sensors for Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets (Junos Telemetry Interface) | 174

Enabling Export of Transit SPRING Statistics | 179

Understanding Enabling Export of Transit SPRING Statistics | 180

Enabling Collection of SPRING Statistics and Exporting Them | 182

Configuring IP-AFT Prefix Filtering | 184

Enable Subscriber and Service Accounting for BNG CUPS Telemetry | 185

CPU and NPU Sensor Support for MX Series Routers with MPC10E-15C-MRATE Line Cards | 189

Delivery of Telemetry Data for AFT-Based Line Cards on MX Series Routers | 189

Diameter Application Protocol And Diameter Peer Sensors For Subscribers | 190

Dynamic Tunnel Statistics Support	191
Enabling Streaming of Telemetry Sensor Information for SR-TE policies (BGP or Static)	192
Exporting Packet Forwarding Engine Traffic Sensor Data	193
Export Timing Data to Collectors	195
FPC and Optics Support	197
Interface Burst Monitoring	198
Interface Express Sensor	198
JTI Broadband Edge Statistics Support for Junos Fusion on MX Series	199
Physical Ethernet Interface Sensor	200
Sensor Power-State Management Support Using gNMI	200
Standby Routing Engine Sensors For Subscribers	203
Support For LSP Statistics	204
NPU Sensors	207
NPU and Firewall Resource Utilization	207
Telemetry Streaming Support for ZR/ZR+ Optics on MPC 10 Linecards	223
Leaf-Level Support for PFE Sensors	224
Resource Filtering	227
Transceiver Diagnostics	233
VLAN Sensors	233
Best Practices for Implementing Junos Telemetry Interface	
Telemetry Resources	236
Guidelines for Specifying Data Reporting Intervals Junos Telemetry Interface	237
Guidelines for Aggregating Junos Telemetry Interface Data	238
Guidelines for Exporting Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets	244

6

Junos Telemetry Interface Plug-ins**Network Telemetry Framework (NTF) Agent | 247**

NTF Agent Overview | 247

Configuring NTF Agent | 248

Open Source Plug-ins | 251

JTI Plug-ins for Open Source Data Collectors | 251

7

Configuration Statements and Operational Commands**Junos CLI Reference Overview | 253**

8

Legacy Information**Legacy Sensor Paths | 255****J-Insight Device Monitor Overview | 335****J-Insight Device Monitor Basic Configuration | 338**

Before you Begin | 338

J-Insight Health Monitoring | 341

J-Insight Fault Monitoring | 341

About This Guide

The Junos Telemetry Interface (JTI) enables you to stream device data from Juniper devices to external data collectors. This data can include information about traffic patterns, device status, error rates, and other metrics that provide insights into the network's health and behaviour. JTI offers solutions for streaming device data over gRPC or UDP connections. JTI connections can originate from a Juniper Device or an external data collector. JTI currently supports a Juniper Native data model and an OpenConfig data model.

Use this guide to understand the concepts and features of the Junos Telemetry Interface, design your telemetry solution, and configure the streaming of device information to external data collectors.

1

PART

Junos Telemetry Interface

- [Understanding Junos Telemetry Interface | 2](#)
-

Understanding Junos Telemetry Interface

SUMMARY

The Junos Telemetry Interface (JTI) is a telemetry framework developed by Juniper Networks. It enables real-time monitoring by exporting operational data from Junos devices to external collectors. This topic describes the concepts of model-driven telemetry, telemetry modes, transport protocols, telemetry sensors, sensor paths, and data models used by JTI.

IN THIS SECTION

- [Network Telemetry | 2](#)
- [Junos Telemetry Interface | 3](#)
- [Model-Driven Telemetry | 3](#)
- [Sensors and Sensor Paths | 4](#)
- [Telemetry Data Collector | 5](#)
- [Telemetry Modes | 5](#)
- [Telemetry Protocols | 7](#)
- [Data Models | 8](#)
- [Explore Sensor Paths | 9](#)
- [Sensor Data Encapsulation Format | 11](#)
- [Native Protocol Buffers \(protobuf\) Message Structure | 12](#)
- [Supported Data Types | 16](#)
- [Performance Monitoring using JTI | 18](#)

Network Telemetry

Network telemetry is the process of collecting, transmitting, and analysing data from network devices. This data can include information about traffic patterns, device status, error rates, and other metrics that provide insights into the health and behaviour of the network. Using this data, you can draw useful insights and apply this information to monitor and manage network performance and security effectively. Network administrators can use telemetry data to troubleshoot network issues, detect anomalies, and optimize resource utilization across the network. One of the key benefits of network telemetry is its ability to provide real-time visibility into network operations.

Network telemetry also plays a crucial role in enhancing network security. By analysing telemetry data, security teams can identify unusual patterns that may indicate a cyber-attack or other security threats, thus enabling faster detection and response to potential security incidents and helping to protect the network and its data.

Junos Telemetry Interface

Junos Telemetry Interface (JTI) is Juniper's telemetry solution, developed to stream telemetry data. The JTI is highly scalable and can support monitoring several devices remotely in a network. It also helps improve troubleshooting, proactively manage the network, and reduce operational costs.

Junos Telemetry Interface can be applied in a variety of network scenarios:

- **Performance monitoring:** Monitor key metrics like interface utilization, latency, and packet loss to ensure optimal network performance.
- **Security monitoring:** Track security events, analyze traffic patterns, and identify potential security threats.
- **Application performance management:** Gain insights into application performance by correlating network data with application data.
- **Network capacity planning:** Analyze historical and real-time data to identify potential bottlenecks and plan for future capacity needs.

Other applications of the JTI include providing real-time data to support operational state synchronization between a network element and an external controller, such as the NorthStar Controller, Juniper's Mist, or Juniper Apstra. The Northstar Controller automates the creation of traffic-engineering paths across the network and subscribes to telemetry data about certain network elements, such as Label Switched Path (LSP) statistics. Apstra is a multivendor automation tool that supports Intent Based Analytics (IBA) that allows you to define how you want to monitor and analyze telemetry data from your network. Juniper Mist Premium Analytics is a cloud-based subscription service that provides end-to-end network observability and business intelligence to IT networking, security, and line-of-business users.

Model-Driven Telemetry

Junos Telemetry Interface has adopted Model Driven Telemetry (MDT) architecture. A model-driven network telemetry system is an advanced approach to network monitoring that leverages data models to define and collect telemetry data from network devices. In this system, data models are often defined using the YANG (Yet Another Next Generation) language to specify the structure and types of data to be collected or streamed.

Juniper currently supports two different data models:

- Juniper Native data model
- OpenConfig data model

Both models use YANG to specify the data structure and data types to be streamed. For more information see, ["Data Models" on page 8](#).

Choosing the right data model depends on the specific needs. You can subscribe to both OpenConfig and native sensors at the same time.

Follow the steps below to set up a model-driven telemetry solution:

1. **Set up the data collector:** Configure network devices to collect and encode the data. For more information, see ["Telemetry Data Collector" on page 5](#).
2. **Establish the transport protocols:** Choose and configure the appropriate transport protocols for data transmission. For more information, see ["Telemetry Protocols" on page 7](#).
3. **Configure the sensor:** A sensor profile defines the parameters of the system resource to monitor and stream data. You can enable only one system resource to monitor 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 more information, see ["Sensors and Sensor Paths" on page 4](#).
4. **Create subscriptions:** Set up subscriptions for the data streams you need to monitor. The telemetry session can be established in dial-in mode or dial-out mode, depending on whether the device or receiver is configured to initiate the subscription. For more information, see ["Telemetry Modes" on page 5](#).

Sensors and Sensor Paths

Telemetry sensors are an important component of a telemetry solution. They measure various physical, environmental, and performance parameters and convert them into signals that are transmitted over gNMI or UDP connections to the collector for remote monitoring and analysis. Some examples are temperature sensors, interface sensors, flow sensors, and so on. A telemetry sensor path is a specific route within a data model, often defined using YANG, that specifies the exact data to be collected and streamed from a network device. Juniper supports both Openconfig and native sensors. Openconfig sensors mostly track counter-based or state-based metrics and native sensors are more effective in tracking event-driven metrics as these sensors have access to in-depth device data. You can configure Openconfig-based sensors paths to retrieve sensor information in a vendor-neutral format or configure native sensor paths to retrieve Juniper proprietary information in a native format. For more information see, ["Explore Sensor Paths" on page 9](#).

Telemetry Data Collector

The telemetry collector is a specialized tool or software that performs data collection, processing, transmission, and storage of telemetry data. It is an intermediary between the Junos devices generating telemetry data and the backend systems that store, analyse, and visualize it.

Functions of a data collector:

- **Data Collection:** The collector receives telemetry data from Juniper devices over gRPC, gNMI, or UDP connections.
- **Data Processing:** The collector processes the collected data by aggregating and normalizing it to filter out unnecessary information, aggregate metrics, and perform an initial analysis. This processing helps reduce the data volume and focus on the most critical metrics.
- **Data Transmission:** It ensures reliable data transmission and low latency.
- **Data Storage:** The processed data is then exported to various backend systems for further analysis, visualization, and storage. It can be stored in data lakes for long-term analysis and historical comparison. The collector supports multiple data formats and protocols, making it compatible with different monitoring and analytics tools. The analysed data can be presented through interactive dashboards and reports, providing actionable insights to network administrators.

Telemetry Modes

The Junos Telemetry Interface supports telemetry sessions in two modes:

- ["Dial-in mode" on page 21](#)
- ["Dial-out mode" on page 88](#)

Juniper devices can operate in both dial-in and dial-out modes. You can configure the Juniper device in either mode based on your network's topology. Both modes use the same data models and stream the same telemetry data over the network. The difference between the two modes is based on whether the collector or Juniper device initiates and maintains the connection.



NOTE: JTI is predominantly a "dial-out" system for streaming telemetry, regardless of whether you're using gRPC (with or without gNMI) or UDP. However, it supports some limited dial-in capabilities.



NOTE: Juniper's gRPC support in JTI is primarily dial-out, where the device acts as the gRPC client and the collector is the gRPC server. Dial-in (collector connecting to the device) isn't a standard JTI feature for gNMI telemetry streaming. However, you can use gNMI in a dial-in fashion for on-demand requests (for example, a one-time "get" operation to fetch data), but this is not the same as a persistent telemetry subscription. For streaming telemetry, gRPC in JTI is usually device-initiated.

Subscription Types

JTI supports various subscription modes, enabling tailored data collection based on specific needs and conditions. The subscription modes are configured on the device and dictate the behavior of the data stream. Implementation of subscription modes depends on whether you're using a dial-out or a dial-in connection. Streaming intervals determine the frequency of telemetry data transmission between the device and the collector. Data collection and streaming are triggered when certain conditions are met or events occur that initiate the collection and streaming of telemetry data. These triggers ensure that data is collected when specific criteria are met. For example, telemetry data can be collected and streamed when packet loss is detected. Both dial-in and dial-out telemetry support the following subscription modes:

- **Once:** This is a one-time request for telemetry data. You can configure this mode when a snapshot of the current state of the subscribed data is required. It is supported in both dial-in and dial-out connections but is primarily used in dial-in connections. The device sends the data once to the collector and stops for a dial-out connection. The collector requests dial-in connections through a "Subscribe" RPC with ONCE mode. In a dial-out connection, you must configure a sensor and subscription to trigger only once, which is not a common scenario. In dial-in connection, it is like a "Get" RPC but framed as a subscription.
- **Poll:** Is periodic on-demand retrieval of telemetry data. This configuration is suitable for monitoring sensors at specific intervals. POLL is supported in a dial-in scenario, where the collector initiates the connection to the device's gNMI server. It's not a typical dial-out streaming mode, as JTI's streaming is device-driven. Requires the device to run a gNMI server. The collector subscribes with POLL and then polls as needed.
- **Stream:** This ongoing subscription continuously streams data when configured triggers occur.

Note: Not all sensors (OpenConfig or native) support all modes.

This form of subscription has three sub-modes:

- **ON_CHANGE:** The device sends updates only when the monitored data changes (for example, an interface counter increments or a state flips). This mode is suitable for event-driven metrics rather than time-driven. Native sensors have an edge over Openconfig sensors in the context of event-driven metrics.

- **SAMPLE:** Telemetry updates are streamed at regular intervals based on the configured interval. This mode is suitable for metrics like packet counts or bytes transferred, which are sampled over time.
- **TARGET_DEFINED:** The device decides the best mode (SAMPLE or ON_CHANGE) based on the monitored sensor or resource. Juniper's implementation may default to SAMPLE unless ON_CHANGE is explicitly supported for the sensor.
 - **NOTE:** The TARGET_DEFINED subscription requests for configuration paths are treated as ON_CHANGE requests only.

Determine the Appropriate Telemetry Configuration for your Network

Guidelines for Configuring the Telemetry Modes

Consider the following guidelines before selecting the telemetry sensors, connection method (dial-in or dial-out), and subscription mode suitable for your network topology:

- A combination of OpenConfig sensors and the subscription mode SAMPLE is ideal for standardized, periodic monitoring (for example, multivendor dashboards).
- A combination of Native sensors and ON_CHANGE subscription mode is suitable for Juniper-specific and event-driven insights (for example, troubleshooting hardware).

To determine the appropriate telemetry session for your network, the following information summarizes a comparison between dial-in and dial-out telemetry modes:

Dial-In vs. Dial-Out

- Dial-in (gRPC with gNMI) pulls snapshots from either sensor type (Openconfig or native).

Example: Collector uses gNMI over gRPC to pull OpenConfig stats (*/interfaces*) or native PFE stats (*/junos/system/linecard*).

- Dial-out (gRPC with gNMI or UDP) streams updates, with gNMI/gRPC favoring structure and UDP suiting native simplicity.

Example: Device streams OpenConfig BGP stats through gNMI over gRPC, or native firewall counters over UDP.

Telemetry Protocols

The Junos Telemetry Interface supports the gNMI and UDP protocols to collect and stream telemetry data from Juniper devices to the data collector.

gRPC Network Management Interface (gNMI)

The gNMI is a protocol based on gRPC that configures and monitors network devices. Network operators can retrieve and modify device configuration data and subscribe to real-time telemetry data from network devices. gNMI supports subscription modes such as ONCE, POLL, and STREAM for telemetry updates. gNMI uses data encoding formats such as Protocol Buffers (protobuf) and ensures secure communication using TLS. For more information, see ["gNMI Service" on page 26](#) and ["Subscribing to Telemetry Data Using gNMI" on page 29](#).

User Datagram Protocol (UDP)

Streaming Telemetry Data over UDP is based on the dial-out mechanism. The sensor paths are configured through the CLI, and the device sends the data for the configured sensor paths over UDP to the collector's destination address. The destination address is configured through the CLI. While streaming telemetry over UDP from native sensors, the data is sent to the collector over UDP in the protobuf format. UDP is suitable for exporting stateless data. For more information, see ["Streaming Telemetry Data Over UDP" on page 89](#).

Data Models

Junos Telemetry Interface data models define the structure of telemetry data collected from network devices, using YANG (Yet Another Next Generation). YANG is a standards-based, extensible data modeling language used in the Junos Telemetry Interface (JTI) to define configuration, operational state data, and remote procedure calls (RPCs) for network devices. In JTI, YANG modules enable the provisioning of sensors to collect and export telemetry data, such as interface statistics, using native or OpenConfig data models. The YANG standard is defined in [RFC 6020](#) and [RFC 7950](#).

Juniper Networks publishes YANG modules for Junos devices, which can be downloaded from the [Juniper GitHub repository](#) or generated on-device.

The OpenConfig working group defines the OpenConfig data model. It is a vendor-neutral data model to configure and manage the network. OpenConfig data model generates data as Google Protocol Buffers (GPB) messages in a universal key/value format. JTI allows you to leverage OpenConfig models for a broader, vendor-agnostic view of your network. Openconfig sensor paths are used to retrieve sensor information from sensors based on the Openconfig data model. For detailed Openconfig resource path exploration, see [Junos YANG Data Model Explorer](#).

The Juniper native data model is an open and extensible framework developed by Juniper. This model is used to stream telemetry data about the unique features found on Juniper devices. These include interface statistics, routing information, security metrics, and so on. Additionally, the native model allows for the definition of enterprise-specific sensors. To access information from Juniper or enterprise-specific sensors, subscribe to Juniper native sensors. Native sensor paths are used to retrieve sensor information from sensors based on the native data model. Juniper's YANG modules for native sensors are available at Juniper's Telemetry [GitHub repository](#).

Explore Sensor Paths

A telemetry sensor path describes the hierarchical path to the data points or metrics that need to be monitored. To stream the required sensor data and activate the sensor and identify the relevant sensor path. JTI supports both Openconfig sensor paths and native sensor paths:

- **Openconfig Sensor Paths**

To configure data collection from Openconfig sensors, define the subscriptions and sensor paths (for example, `/interfaces/interface/state/counters`), set up the data collector, and download the Junos Telemetry Interface protocol buffers files from the Juniper Networks support page. Capture and decode the captured data on the collector.

- **Native Sensor Paths**

Native sensor paths are specific to Junos OS (for example, `/junos/system/line card/interface/`) and offer granular, Juniper-optimized access to device-specific metrics.

To configure data collection from native sensors, use the Junos CLI to provision native sensors for collecting specific data. Configure the telemetry interface to stream data using gRPC or UDP. Use the protocol buffer's files to decode the streamed data on the collector.

Both path types enable the structured output of data in formats such as JSON or XML, ensuring compatibility with external collectors for efficient monitoring and analytics.

Sensor Path Explorer

The Juniper Networks [Junos YANG Data Model Explorer](#) is an online tool for viewing all the supported resource paths, their corresponding leaves, and the device platforms that support them. It enables you to explore or compare various OpenConfig and Native data model attributes. Use the filter option based on the software release number or product to view the list of resource paths and sensors on each platform.



NOTE: The

[Junos YANG Data Model Explorer](#)

was introduced in the 23.2R2-S2 releases. From releases 20.2R1 up to 23.1R1, the sensor information is available in the

[Junos Telemetry Sensor Explorer](#)

.

Selecting Telemetry Sensor Paths

In a model-driven telemetry system, the sensor path can be configured to end at any level within the data model's container hierarchy. Based on the required telemetry information, you can configure the sensor path to retrieve a broad data set or be very specific and retrieve targeted information for a particular sensor. For example, a sensor path might point to a container that includes all interface statistics on a router, or it could be more granular, focusing on a single metric like packet loss on a specific interface.

For example, to receive telemetry data about alarms generated on the device (using the OpenConfig data model), you can configure either of the following resource paths based on the granularity of sensor data required:

- `/system/alarms/alarm/id`: This path retrieves only the alarm ID.
- `/system/alarms/alarm/config`: This path retrieves the detailed alarm information.

Configuring the correct sensor paths ensures an efficient telemetry system. Each resource path enables data streaming for the system resource globally, that is, systemwide. You can modify each resource path to specify a logical or physical interface. The resource path `/interfaces/interface/config` retrieves the list of configurable items at the global, physical interface level, whereas the path `/interfaces/interface/config/name` specifies the name of the interface, and the device may restrict the allowed values for this leaf depending on the type of the interface.

Important Guidelines for Selecting Sensor Paths

- Users should always provide the complete and direct resource path when configuring sensors. Providing partial resource paths, such as `/components/component/`, results in incomplete configurations and potential errors. Such resource paths overwhelm the device, as it needs to display all the available options at that hierarchy. To prevent this, always verify and use the full resource path to ensure precise and efficient sensor configuration.



NOTE: Creating subscription and sensor configuration at the `/` (root) and `/junos/` is not allowed.

Table 1: Sensor Path Example

Good example of a sensor path	Poor example of a sensor path
/interfaces/interface/subinterfaces/subinterface/ state/counters/out-pkts	/interfaces/interface

- The logical and physical Packet Forwarding Engine interface sensors report some leaves inconsistently to the collector. For example, the subscribed path `/interfaces/ 115 interface/` producing the streamed path `/junos/system/linecard/interface/logical/ usage/` reports key name leaves `parent_ae_name` and `init_time` (with underscores in the leaf name). The subscribed path `/interfaces/interface/state/` producing the streamed path `/ junos/system/linecard/interface/queue/` reports key name leaves `parent-ae-name` and `init\u0002time` (with hyphens in the leaf name).

Sensor Data Encapsulation Format

The Junos telemetry interface supports two ways of exporting data in the protocol buffers (gpb) format. This section describes the format of data exported from native sensors 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.

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 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 ["Explore Sensor Paths" on page 9](#). For a a list of supported native sensors, see *sensor (Junos Telemetry Interface)*.

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: <https://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)*.

Native Protocol Buffers (protobuf) Message Structure

The information streamed to the collector is sent using the top-level message structure of TelemetryStream (*telemetry_top.proto* file). The message contains the metadata of the sensor data being streamed (for example, sensor path, the system from where data is being sent, the node from where data is sent, and so on). The actual sensor data is sent as an extension to the top-level message. A separate proto file is sent for the sensor data. The *telemetry_top.proto* file and the sensor proto file are used to decode the sensor data at the collector.

Structure of *telemetry_top.proto* File

```
message TelemetryStream {
    required string system_id = 1
    [(telemetry_options).is_key = true];
    optional uint32 component_id = 2
    [(telemetry_options).is_key = true];
    optional string sensor_name = 4
    [(telemetry_options).is_key = true];
    ...
    optional EnterpriseSensors enterprise = 101;
}
message EnterpriseSensors {
    extensions 1 to max;
}
extend EnterpriseSensors {
    // re-use IANA assigned numbers
    optional JuniperNetworksSensors juniperNetworks = 2636;
}
message JuniperNetworksSensors {
    extensions 1 to max; → ends with the extension message
}
```



NOTE: This file ends with the extension message.

Structure of Sensor Proto File

This file begins with the extension message.

```

extend JuniperNetworksSensors { → begins with the extension message
    optional Port jnpr_interface_ext = 3;
}
message Port {
    repeated InterfaceInfos interface_stats = 1;
}
message InterfaceInfos {
    required string if_name = 1
    [(telemetry_options).is_key = true];
    optional uint64 init_time = 2;
    ....
}

```

The `TelemetryStream` message also includes optional nested structures that carry different types of data. The nested structures can also carry privately defined sensor data such as `EnterpriseSensors`. See the example below:

```

//
// 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;  
}
```

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.



NOTE: Recommended: 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 2 on page 15](#) for descriptions of each element collected by sensor data, including semantics and corresponding schema.

Table 2: 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.
Average	The average of several samples of a base metric. For example, an <i>average queue depth</i> 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.

Table 2: Individual Data Element Types in the gpb Message (*Continued*)

Element Type	Description
Peak	Maximum value among several samples of a base metric. For example, a <i>peak queue depth</i> 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.

Supported Data Types

A GetRequest is sent when a collector client initiates a Get RPC to receive telemetry data. Specified within the GetRequest are the data elements with which the target should return data to the collector, including the data type. The data type is the variable that specifies the form in which data should be delivered.

Table 3 on page 16 lists the data types supported with Junos telemetry interface (JTI). Unless specified, the data type is supported for JTI data export using remote procedure call (gRPC) services, gRPC Network Management Interface (gNMI) services, or through UDP.

Table 3: Data Types

Type	Value	Description
string	string_val = 1	String value.
int64	int_val = 2	Integer value.
uint64	uint_val = 3	Unsigned integer value.

Table 3: Data Types *(Continued)*

Type	Value	Description
bool	bool_val = 4	Bool value.
float	float_val = 6	Floating point value. NOTE: This data type is deprecated. Use double_val instead of this data type.
double	double_val	A double value is a 64-bit floating point value.
Decimal64	decimal_val = 7)	Decimal64 encoded value. Supported only with gNMI services. Use decimal64 to encode a fixed precision decimal number. The value is expressed as a set of digits with the precision specifying the number of digits following the decimal point in the digit set. For example: <pre>message Decimal64 { int64 digits = 1; // Set of digits. uint32 precision = 2; // Number of digits following the decimal point.</pre> NOTE: This data type is deprecated. Use double_val instead of this data type.
ScalarArray	leaflist_val = 8	Mixed type scalar array value. An homogenous array of the values of mixed datatypes (string, int64, uint64, bool float or decimal64). Supported only with gNMI services.

Table 4: Open Config Data Type

Type	Value	Description
ieeefloat32	Binary, length = 4	<p>An IEEE 32-bit floating point number. This is a Open Config model data type.</p> <p>The format of this number is of the form:</p> <pre> 1-bit sign 8-bit exponent 23-bit fraction </pre> <p>The floating point value is calculated using:</p> $(-1)^{sign} * 2^{(Exponent-127)} * (1 + \frac{Fraction}{2^{23}})$ <p>NOTE: The gNMI data type equivalent to binary format is "byte." In a gNMI response, data was incorrectly received from a device in float_val format, and a non-compliance error was displayed. Changes have been incorporated to return data in the bytes_val format. The four bytes (that represent a floating-point value) are sent in network byte order. Ensure to reorder them in host byte order before interpreting the value.</p>

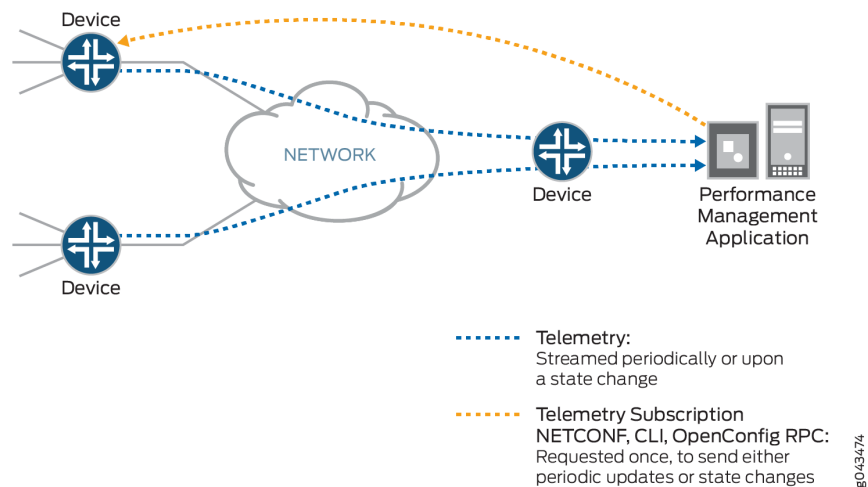
For more information on data types, see [github](#) and <http://www.openconfig.net/>.

Performance Monitoring using JTI

One 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.

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 19](#) 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



Starting in Junos OS Release 18.1R1, a new sensor is available that allows syslog data to be streamed to network telemetry collector systems. Using the `/junos/events/` sensor, and an export profile with a reporting-rate of 0, you can now stream event data along with statistical data to your telemetry-collection systems.

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.

2

PART

Dial-in Telemetry

- Understanding Dial-in Telemetry | **21**
 - Guidelines for Telemetry Data Subscriptions over gNMI | **29**
 - Configure gRPC Services | **51**
 - Establish a Dial-in Telemetry Connection | **71**
-

Understanding Dial-in Telemetry

SUMMARY

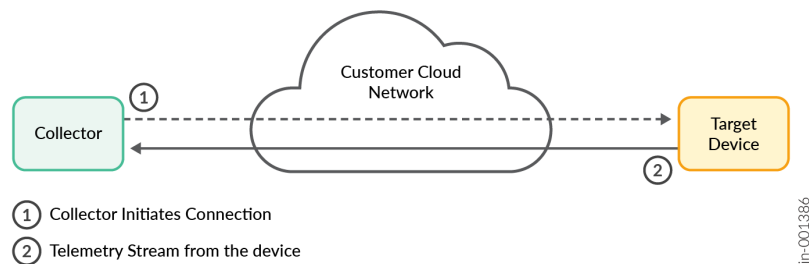
Dial-in telemetry is a network monitoring method where the collector initiates the connection to the device to retrieve telemetry data. The collector "dials in" to the network device, typically using protocols like gRPC to request and receive data.

IN THIS SECTION

- [gRPC Service | 22](#)
- [gRPC Server Overview | 25](#)
- [gNMI Service | 26](#)

In dial-in mode, the data collector initiates the connection to the network device and subscribes to telemetry data. A session is established between the collector and the device, and then data is streamed from the device to the collector at configured intervals. This method is preferred by operators who want a single configuration and operational data channel.

Figure 2: Dial-in Telemetry



The JTI supports both gNMI and Juniper Extension Toolkit (JET) dial-in connections over gRPC transport.



NOTE: Juniper's gNMI support in JTI is primarily dial-out, where the device acts as the gRPC client and the collector is the gRPC server. Dial-in (collector connecting to the device) isn't a standard JTI feature for gNMI/gRPC telemetry streaming. However, you can use gNMI in a dial-in fashion for on-demand requests (for example, a one-time "get" operation to fetch data), but this is not the same as a persistent telemetry subscription. For streaming telemetry, gRPC in JTI is usually device-initiated. For gNMI dial-in

connections gNMI service must be enabled. The POLL (collector polls snapshots), and ONCE (single snapshot) subscription modes are supported.

gRPC Service

IN THIS SECTION

- [Using gRPC to Stream Data | 22](#)
- [Enable Junos Telemetry Interface Using OpenConfig | 24](#)

gRPC is an open-source framework that provides secure and reliable data transport. 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. gRPC remote procedure calls (gRPC) are used to provision sensors and to subscribe to and receive telemetry data. OpenConfig supports the YANG data models. The OpenConfig data model generates data as Google Protocol Buffer (gpb) messages in a universal key/value format.



NOTE: Starting in Junos OS Release 18.2R1, OpenConfig-based routing engine (RE) sensors can stream data as gpb-structured messages over UDP.

Using gRPC to Stream Data

According to the OpenConfig specification, only gRPC-based transport is supported for streaming data. The gRPC server 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 .



NOTE: Starting in Junos OS Release 18.2R1, when an external streaming server, or collector, provisions sensors to export data through gRPC on devices running Junos OS, the sensor configuration is committed to the `junos-analytics` instance of the ephemeral configuration database, and the configuration can be viewed by using the `show ephemeral-configuration instance junos-analytics operational` command. In earlier releases, the sensor

configuration is committed to the default instance of the ephemeral configuration database.



NOTE: The Juniper telemetry header that was exported as part of updates is now exported as an extension header. `GnmiJuniperTelemetryHeader.proto` is used to decode the updates from Juniper devices running Junos OS Release 19.3 or earlier and `GnmiJuniperTelemetryHeaderExtension.proto` is used for devices running Junos OS Release 19.4 or later.

See [Table 5 on page 23](#) for a list and descriptions of the RPCs implemented to support the Junos telemetry interface.

Table 5: 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 <code>telemetrySubscribe</code> .
cancelSubscription	Unsubscribe a subscription created through <code>telemetrySubscribe</code> .

Data streamed through gRPC is formatted in OpenConfig key/value pairs in 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 as interface counters, and the state of the resource.



NOTE: Starting in Junos OS Release 18.2R1, data streamed through gRPC can be formatted as protobuf in addition to key/value pairs for OpenConfig-based routing engine (RE) sensors. These sensors are in addition to the packet forwarding engine (PFE) sensors.

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"
```

A mapping table maps field names to the OpenConfig key strings.

Enable Junos Telemetry Interface Using OpenConfig

OpenConfig for Junos OS specifies an RPC model to enable the Junos telemetry interface. This package also includes the required YANG models.

You can locate all [YANG data models](#) in a GitHub repository for a given OS and release in a single download package. The package and repository include the native configuration, state, and RPC data models and the OpenConfig and IETF data models supported by that OS. You can also access the YANG data models from the Juniper Networks download site.

Using a Web browser, navigate to the All Junos Platforms software download URL on the Juniper Networks webpage: <https://www.juniper.net/support/downloads/>. From the Network Management tab, scroll down to select OpenConfig. Select the Software tab. Select the appropriate version of OpenConfig module.

The programmatic interface `OpenConfigTelemetry` 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 for data at the specified path. The device should then send telemetry data back on the same connection as the subscription request.

gRPC Server Overview

SUMMARY

The Junos Telemetry Interface (JTI) includes the capability for multi-port service configuration, allowing you to configure multiple sets of telemetry services to listen on different ports.

The Telemetry Interface introduces flexible gRPC service configuration capabilities, enabling you to set up multiple gRPC servers, each with distinct services, listening addresses, and ports. This functionality provides granular control over service management and telemetry data collection. You can configure TLS certificates for each server to ensure secure communications. To configure gRPC server, see *Configure gRPC Services*.

gNMI Service

IN THIS SECTION

- [gNMI Origin | 27](#)

The gNMI (gRPC Network Management Interface) is a protocol based on gRPC that configures and monitors network devices. Developed by OpenConfig specifically for network management, gNMI allows network operators to retrieve and modify device configuration data and subscribe to real-time telemetry data from network devices. Data collection is a key task in telemetry solutions. gNMI supports subscription modes such as ONCE, POLL, and STREAM for telemetry updates. gNMI leverages the gRPC framework and uses efficient data encoding formats such as Protocol Buffers (protobuf), and assures secure communication using TLS. YANG models to define structure of configuration and telemetry data.

The main components of gNMI are:

- **gNMI Client:** Runs on an external system. It sends gNMI requests to configure the device, retrieve configuration data, and subscribe to telemetry streams.
- **gNMI Server:** Runs on the network device and provides access to telemetry and configuration data. The gNMI server processes the request based on its type. If it is a configuration management change, it updates the device configuration and sends a response to the client. The data will be streamed to the gNMI client if it is a telemetry data collection request.

gNMI supports the following Remote Procedure Calls (RPCs) for managing and monitoring network devices:

- **Get:** This RPC retrieves the current state of the network device, including configuration and operational data.



NOTE: In the gNMI get command, the `type` option is used to specify the type of data to be retrieved. The available options are `CONFIG`, `STATE`, and `ALL`. The valid type option is `CONFIG`. Encoding defines the value encoding formats that are supported by the gNMI protocol. `JSON`, `BYTES`, `PROTO`, `ASCII`, and `JSON_IETF` are available options. The valid encoding options are `JSON_IETF` and `ASCII`. Choosing any other configuration apart from the valid options will result in an error message.

- **Set:** The set RPC modifies the configuration of the network device.

- **Subscribe:** You can use this RPC to subscribe to telemetry data from a network device. The following subscription modes are supported:
 - **ONCE:** Retrieves the current values only once.
 - **POLL:** Sends current values whenever a poll message is received.
 - **STREAM:** Continuously sends updates at specified intervals or when changes occur.

For more information, see ["Guidelines for Telemetry Data Subscriptions over gNMI" on page 29](#)

- **Capabilities:** This RPC discovers the capabilities of the network device, such as supported models and encodings.

gNMI Origin

The `origin` field in the `Path` message identifies a schema that the path belongs to. `origin` is encoded as a string. The path specified within the message is uniquely identified by the tuple of `<origin, path>`.

The `origin` field is valid in any context of a `Path` message. Typically it is used in the following ways:

- In a `SetRequest` to indicate a particular schema is being used to modify the target configuration.
- In a `GetRequest` to retrieve the contents of a particular schema, or in a `GetResponse` to indicate that the payload contains data from a particular `<origin, path>` schema.
- In a `SubscribeRequest` to subscribe to paths within a particular schema, or `SubscribeResponse` to indicate an update corresponds to a particular `<origin, path>` tuple.

If more than one `origin` is used within a message, a path in the prefix must not be specified, because a prefix applies to all paths within the message. In the case that a prefix is specified, it must specify any required `origin`. A single request must not specify `origin` in both prefix and path fields in any RPC payload messages.

Special Values of Origin

Origin values are agreed out-of-band to the gNMI protocol. Where the `origin` field is unspecified, its value should default to `openconfig`. It is recommended that the `origin` is explicitly set.

Definition of origin for YANG-modelled Data

The `openconfig-extensions:origin` field may be utilised to determine the origin within which a particular module is instantiated.



NOTE: `origin` is distinct from `namespace`. While a YANG namespace is defined at any depth within the schema tree, an `origin` is only used to disambiguate entire schema trees. That is to say, any element that is not at the root inherits its `origin` from its root entity, regardless of the YANG schema modules that make up that root.

Partial Specifications of Origin in Set

If a Set RPC specifies `delete`, `update`, or `replace` fields which include an `origin` within their Path messages, the corresponding change must be constrained to the specified `origin` in the following ways:

- `replace` operations must only replace the contents of the specified `origin` at the specified path. Origins that are not specified within the `SetRequest` must not have their contents replaced. In order for a `replace` operation to replace any contents of an `origin` it must be explicitly specified in the `SetRequest`.
- `delete` operations must delete only the contents at the specified path within the specified `origin`. To delete contents from multiple origins, a client must specify multiple paths within the `delete` of the `SetRequest`.

These rules apply where origins represent data that doesn't overlap. in some cases (e.g., CLI and OpenConfig) origins may reflect different 'views' on the same data, and thus their interaction is more complex.

Transactionality of Sets with multiple Origins

Where a `SetRequest` specifies more than one `origin` - i.e., two or more operations whose path include more than one `origin` - manipulations to all affected trees must be considered as a single transaction. That is to say, only if all transactions succeed should the `SetResponse` indicate success. If any of the transactions fail, the contents of all origins must be rolled back, and an error status returned upon responding to the Set RPC.

Guidelines for Telemetry Data Subscriptions over gNMI

SUMMARY

This section describes the subscription modes supported by gNMI connections.

IN THIS SECTION

- [Examples | 35](#)

The gNMI protocol defines the `Subscribe` RPC for subscribing to telemetry data. The telemetry collector uses this RPC to request updates from the network device for state and configuration data.

Requests for new subscriptions are encapsulated within a `SubscribeRequest` message containing one or more resource paths. The subscribed paths relate to specific data instances on the target network device. The request can contain paths based on the OpenConfig or native Junos schemas.

The subscription request must also include one of the following modes:

- `ONCE` - a one-time request for data.
- `POLL` - for periodic, on-demand retrieval of data.
- `STREAM` - a long-lived subscription that streams data according to specified triggers.

Subscriptions in `STREAM` mode must specify one of the following sub-modes:

- `ON_CHANGE` - data updates are only sent when the value of the data item changes.
- `SAMPLE` - data updates are sent once per sample interval based on an interval period specified in the subscription request. The default sample interval is 30 seconds.
- `TARGET_DEFINED` - the network device receiving the subscription request determines the best type of delivery for the data on a per-leaf basis. If the path specified within the message refers to event-driven data, then an `ON_CHANGE` subscription might be created. For data that represents counter values a `SAMPLE` subscription might be created.



NOTE: The `TARGET_DEFINED` subscription requests for configuration paths are treated as `ON_CHANGE` requests only.

For ONCE, ON_CHANGE and SAMPLE subscriptions, the collector can request an initial update containing the current state of the paths in the subscription. This update, also known as an initial sync, is valuable because:

- The collector has a complete view of the current state of every field on the device for that sensor path.
- Event-driven data (ON_CHANGE) is received by the collector at least once before the next event is seen. In this way, the collector is aware of the data state before the next event happens.
- Packet Forwarding Engine sensors that contain zero counter values that normally do not show up in streamed data due to zero-suppression are sent. This ensures that all fields from each line card are known to the collector.

The target device responds to the subscription request with a `SubscribeResponse` message. If the subscription request calls for an initial sync, the target sends the data followed by the response message with the `sync_response` flag set to `true`. After the initial sync, the target device proceeds with updates for the paths according to the subscription mode.

The `SubscribeRequest` message includes a flag called `updates_only`. When this flag is set to `true`, the target device does not send an initial sync, only subsequent updates as follows:

- For STREAM subscriptions in SAMPLE mode, the update is sent at the next sample interval.
- For STREAM subscriptions in ON_CHANGE mode, the update is sent upon the next value change.
- For ONCE subscriptions, only the `SubscribeResponse` is sent with `sync_response` set to `false`, and the subscription is closed.
- TARGET_DEFINED subscriptions are treated as ON_CHANGE for configuration paths and the update is sent upon the next value change.

The contents of the `SubscribeRequest` and `SubscribeResponse` messages are defined in [gnmi.proto](#) file. For more information on the Subscribe RPC and subscription modes, see the gNMI specification at: [gNMI Specification: Subscribing to Telemetry Updates](#).



NOTE: The following limitations apply to configuration paths:

- POLL subscriptions are not supported.
- Prefix paths are not included in the update messages.

- The gNMI response is not supported for Juniper-specific metadata operations, such as active/inactive, insert before/after, comment/annotate, and protect/unprotect. These might appear in the message but are not valid.
- Unsupported parameters in the SubscribeRequest include `suppress_redundant`, `heartbeat_level`, `allow_aggregation`, and `qos`.
- Only PROTO encoding is supported.
- Extensions in SubscribeRequest messages are not supported
- Filtering of the subscription paths is supported only at a key level.
- The following commit variants are not supported for ON_CHANGE and TARGET_DEFINED subscriptions:

`commit at`, `commit prepare/activate`, and `batch commits`.
- Commits are not supported from the

`edit dynamic` and `edit private` edit or configure modes.
- Update messages are not sent for presence containers.
- For subscription lists where the identifier or key is the only leaf configured, there might not be an update message.

Enabling “ON CHANGE” Sensor Support Through gNMI

Periodical streaming of OpenConfig operational states and counters has been supported since Junos OS Release 16.1, exporting telemetry data from Juniper equipment to an external collector. While useful in collecting all the needed information and creating a baseline “snapshot,” periodical streaming is less useful for time-critical missions. In such instances, you can configure ON_CHANGE streaming for an external collector to receive information only when operational states experience a change in state.

To support ON_CHANGE streaming, a new specification called gRPC Network Management Interface (gNMI) is implemented for the modification and retrieval of configurations from a network element. Additionally, the gNMI specification can be used to generate and control telemetry streams from a network element to a data collection system. Using the new gNMI specification, one gRPC service definition can provide a single implementation on a network element for both configuration and telemetry as well as a single NMS element to interact with a device by means of telemetry and configuration RPCs.

The Junos file package (junos-telemetry-interface) includes the gnmi.proto file and GnmiJuniperTelemetryHeader.proto Juniper extension for gNMI support.

Information about the RPCs supporting this feature can be found in the gNMI Proto file version 0.4.0 (the supported version) and the specification released

- [gNMI Specification: STREAM Subscription](#)
- [gNMI Specification: gNMI Protocol](#)

The telemetry RPC `subscribe` under gNMI service supports `ON_CHANGE` streaming. RPC `subscribe` allows a client to request the target to send it values of particular paths within the data tree. Values may be streamed (`STREAM`), sent one-off on a long-lived channel (`POLL`), or sent one-off as a retrieval (`ONCE`).

If a subscription is made for a top level container with a sample frequency of 0, leaves with `ON_CHANGE` support are streamed based on events. Other leaves will not be streamed.



NOTE: In order to permit a device to decide which nodes will be streamed as `ON_CHANGE` and which will `SAMPLE`, the collector must subscribe for `TARGET_DEFINED` with `sample_interval`.

Enabling “TARGET_DEFINED” Subscription Mode through gNMI

Starting with Junos OS Release 20.2R1, `TARGET_DEFINED` subscription mode with gRPC Network Management Interface (gNMI) services is supported for JTI on MX5, MX10, MX40, MX80, MX104, MX150, MX204, MX240, MX480, MX960, MX2008, MX2010, MX2020, MX10003, MX10008, and MX10016 routers.

Using a gNMI subscription, an external collector stipulates how sensor data should be delivered:

- `STREAMING` mode periodically streams sensor data from the DUT at a specified interval.
- `ON_CHANGE` mode sends updates for sensor data from the DUT only when data values change.
- Newly supported `TARGET_DEFINED` mode (submode 0) instructs the DUT to select the relevant mode (`STREAMING` or `ON_CHANGE`) to deliver each element (leaf) of sensor data to the external collector. When a subscription for a sensor with submode 0 is sent from the external collector to the DUT, the DUT responds, activating the sensor subscription so that periodic streaming does not include any of the `ON_CHANGE` updates. However, the DUT will notify the collector whenever qualifying `ON_CHANGE` events occur.

Subscriptions will default to a periodic streaming frequency of 30 seconds unless otherwise specified by the collector in the subscription request.

The Java Script Object Notification (JSON) file below shows a sample gNMI subscription. TARGET_DEFINED mode is set using `submode=0` for the resource (sensor) path `/interfaces/interface[name='lo0']/state`.

```
$ cat gnmi.json
{
  "dut_list":[
    {
      "port":32767,
      "rpc":["sub_request"],
      "sub_request":{
        "subscription":[
          {
            "path":"/interfaces/interface[name='lo0']/state",
            "submode":0,
            "sample_interval":30
          }
        ],
        "mode":0,
        "encoding":2
      }
    }
  ]
}

$ python ./gnmi_subscribe_client_sample.py -c ./gnmi.json -d 10.53.32.102 -l client.log
```

The Junos file package (`junos-telemetry-interface`) includes the `gnmi.proto` file and `GnmiJuniperTelemetryHeader.proto` Juniper extension for gNMI support.

For more information, see the gNMI specifications and gNMI protocol file here:

- [gNMI Specification: STREAM Subscription](#)
- [gNMI Specification: gNMI Protocol](#)

Enabling “INITIAL_SYNC” Subscription Mode through gNMI

Starting in Junos OS Release 20.2R1, INITIAL_SYNC statistics from Packet Forwarding Engine sensors using gNMI services on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches is supported.

Starting in Junos OS Evolved Release 20.4R1, INITIAL_SYNC statistics from Packet Forwarding Engine sensors using gNMI services on QFX5130-32CD switches is supported.

When an external collector sends a subscription request for a sensor with INITIAL_SYNC (gnmi-submode 2), the host sends all supported target leaves (fields) under that resource path at least once to the collector with the current value. This is valuable because:

- The collector has a complete view of the current state of every field on the device for that sensor path.
- Event-driven data (ON_CHANGE) is received by the collector at least once before the next event is seen. In this way, the collector is aware of the data state before the next event happens.
- Packet Forwarding Engine sensors that contain zero counter values (zero-suppressed) that normally do not show up in streamed data are sent, ensuring that all fields from each line card (also referred to as source) are known to the collector.

INITIAL_SYNC submode requires that at least one copy to be sent to the collector; however, sending more than one is acceptable.

Subscriptions will default to a periodic streaming frequency of 30 seconds unless otherwise specified by the collector in the subscription request.

The Java Script Object Notification (JSON) file below shows a sample gNMI subscription. INITIAL_SYNC mode is set using `gnmi_submode 2` for the resource (sensor) path `/interfaces/`. The `gnmi_mode` is set to 0. The protocol encoding is set to 2 for GBP.

```
{
  "influx": {
    "server": "server1",
    "port": 8086,
    "dbname": "gD40",
    "measurement": "OC",
    "user": "influx",
    "password": "influxdb",
    "recreate": true
  },
  "gnmi": {
    "mode": 0,      <---- STREAM
    "encoding": 2, <--- PROTO encoding
    "prefix": "/x/y/z"
  },
  "host": "10.10.130.73",
  "port": 10162,
  "user": "user1",
```

```

    "password": "password1",
    "cid": "cid-1jk",
    "paths": [
      {
        "path": "/interfaces/",
        "Freq": 1000000000,
        "gnmi_submode": 2    <----  SAMPLE
      }
    ]
  }
}

```

The Junos file package (junos-telemetry-interface) includes the gnmi.proto file and GnmiJuniperTelemetryHeader.proto Juniper extension for gNMI support.

For more information, see the gNMI specifications and gNMI protocol file here:

gNMI telemetry specification gNMI protocol definition

- [gNMI Specification: STREAM Subscription](#)
- [gNMI Specification: gNMI Protocol](#)

Examples

IN THIS SECTION

- [Example: ONCE mode | 36](#)
- [Example: ON_CHANGE | 39](#)
- [Example: SAMPLE | 43](#)

The following examples show subscription requests and responses made by a gNMI client and target device in protobuf format.

Example: ONCE mode

The following example shows a subscription request sent from a gNMI client in protobuf format. The subscription mode is ONCE and the OpenConfig resource path is **/system/aaa/authentication/users**:

```
root@controller:~# /usr/local/bin/gnmic sub -a 10.225.0.0:32767 --mode once --path /system/aaa/
authentication/users -u <username> -p <password> --format prototext
```

The target responds with a one-time update:

```
update: {
  timestamp: 1676294840
  update: {
    path: {
      elem: {
        name: "system"
      }
      elem: {
        name: "aaa"
      }
      elem: {
        name: "authentication"
      }
      elem: {
        name: "users"
      }
      elem: {
        name: "user"
        key: {
          key: "username"
          value: "test1"
        }
      }
      elem: {
        name: "config"
      }
      elem: {
        name: "username"
      }
    }
  }
  val: {
```

```

        string_val: "test1"
    }
}
update: {
    path: {
        elem: {
            name: "system"
        }
        elem: {
            name: "aaa"
        }
        elem: {
            name: "authentication"
        }
        elem: {
            name: "users"
        }
        elem: {
            name: "user"
            key: {
                key: "username"
                value: "test1"
            }
        }
        elem: {
            name: "config"
        }
        elem: {
            name: "password"
        }
    }
    val: {
        string_val: "$ABC123"
    }
}
update: {
    path: {
        elem: {
            name: "system"
        }
        elem: {
            name: "aaa"
        }
    }
}

```

```

    elem: {
      name: "authentication"
    }
    elem: {
      name: "users"
    }
    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test1"
      }
    }
  }
  elem: {
    name: "config"
  }
  elem: {
    name: "role"
  }
}
val: {
  string_val: "superuser"
}
}
update: {
  path: {
    elem: {
      name: "system"
    }
    elem: {
      name: "aaa"
    }
    elem: {
      name: "authentication"
    }
    elem: {
      name: "users"
    }
    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test2"
      }
    }
  }
}

```

```

    }
  }
  elem: {
    name: "config"
  }
  elem: {
    name: "role"
  }
}
val: {
  string_val: "superuser"
}
}
}

```

Example: ON_CHANGE

The following example shows a subscription request in STREAM mode with ON_CHANGE sub-mode. The OpenConfig resource path is `/system/aaa/authentication/users/user[username="test1"]`:

```

root@controller:~# /usr/local/bin/gnmic sub -a 10.225.0.0:32767 --mode stream --stream-mode on-
change --path /system/aaa/authentication/users -u <username> -p <password> --format prototext

```

The OpenConfig configuration at the time of the subscription request:

```

user@root> show configuration openconfig-system:system aaa authentication | display set
set openconfig-system:system aaa authentication users user test1 config password $ABC123
set openconfig-system:system aaa authentication users user test1 config role superuser

```

The example response message shows the values for the configuration paths and the `sync_response` flag set to true:

```

update: {
  timestamp: 1676311979
  update: {
    path: {
      elem: {
        name: "system"
      }
    }
  }
}

```

```

    elem: {
      name: "aaa"
    }
    elem: {
      name: "authentication"
    }
    elem: {
      name: "users"
    }
    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test1"
      }
    }
    elem: {
      name: "config"
    }
    elem: {
      name: "password"
    }
  }
  val: {
    string_val: "$ABC123"
  }
}
update: {
  path: {
    elem: {
      name: "system"
    }
    elem: {
      name: "aaa"
    }
    elem: {
      name: "authentication"
    }
    elem: {
      name: "users"
    }
    elem: {
      name: "user"
    }
  }
}

```



```

    key: {
      key: "username"
      value: "test1"
    }
  }
  elem: {
    name: "config"
  }
  elem: {
    name: "role"
  }
}
val: {
  string_val: "superuser"
}
}
}

sync_response: true

```

The following configuration changes are made to the subscribed paths:

- Add username for test1.
- Delete password for test1.

The target device sends the following update in response:

```

update: {
  timestamp: 1676312428
  update: {
    path: {
      elem: {
        name: "system"
      }
      elem: {
        name: "aaa"
      }
      elem: {
        name: "authentication"
      }
      elem: {
        name: "users"
      }
    }
  }
}

```

```

    }
    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test1"
      }
    }
  }
  elem: {
    name: "config"
  }
  elem: {
    name: "username"
  }
}
val: {
  string_val: "test1"
}
}
delete: {
  elem: {
    name: "system"
  }
  elem: {
    name: "aaa"
  }
  elem: {
    name: "authentication"
  }
  elem: {
    name: "users"
  }
  elem: {
    name: "user"
    key: {
      key: "username"
      value: "test1"
    }
  }
}
  elem: {
    name: "config"
  }
  elem: {

```

```

        name: "password"
    }
}
}

```

Example: SAMPLE

The following example shows a subscription request in STREAM mode and SAMPLE sub-mode. The OpenConfig resource path is `/system/aaa/authentication/users/user[username="test1"]`:

```

root@controller:~# /usr/local/bin/gnmic sub -a 10.225.0.0:32767 --mode stream --stream-mode
sample --sample-interval 5s --path /system/aaa/authentication/users -u <username> -p <password>
--format prototext

```

The OpenConfig configuration at the time of the subscription request:

```

user@root> show configuration openconfig-system:system aaa authentication | display set
set openconfig-system:system aaa authentication users user test1 config username test1
set openconfig-system:system aaa authentication users user test1 config password "$ABC123"
set openconfig-system:system aaa authentication users user test1 config role superuser

```

The example response messages show an initial update sent with the `sync_response` flag set to true and subsequent updates sent at 5 second intervals:

```

update: {
  timestamp: 1676295454
  update: {
    path: {
      elem: {
        name: "system"
      }
      elem: {
        name: "aaa"
      }
      elem: {
        name: "authentication"
      }
      elem: {
        name: "users"
      }
    }
  }
}

```

```

    }
    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test1"
      }
    }
  }
  elem: {
    name: "config"
  }
  elem: {
    name: "username"
  }
}
val: {
  string_val: "test1"
}
}
update: {
  path: {
    elem: {
      name: "system"
    }
    elem: {
      name: "aaa"
    }
    elem: {
      name: "authentication"
    }
    elem: {
      name: "users"
    }
    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test1"
      }
    }
  }
  elem: {
    name: "config"
  }
}

```

```
    elem: {
      name: "password"
    }
  }
  val: {
    string_val: "$ABC123"
  }
}
update: {
  path: {
    elem: {
      name: "system"
    }
    elem: {
      name: "aaa"
    }
    elem: {
      name: "authentication"
    }
    elem: {
      name: "users"
    }
    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test1"
      }
    }
    elem: {
      name: "config"
    }
    elem: {
      name: "role"
    }
  }
  val: {
    string_val: "superuser"
  }
}
}
```

sync_response: true

```

update:
{

    timestamp:
1676295459

    update: {
        path: {
            elem:
{

                name:
"system"

            }
            elem: {
                name: "aaa"
            }
            elem: {
                name: "authentication"
            }
            elem: {
                name: "users"
            }
            elem:
{

                name: "user"
                key: {
                    key: "username"
                    value: "test1"
                }
            }
            elem: {
                name: "config"
            }
            elem: {
                name: "username"
            }
        }
        val: {
            string_val: "test1"

```

```

    }
  }
  update: {
    path: {
      elem: {
        name: "system"
      }
      elem: {
        name: "aaa"
      }
      elem: {
        name: "authentication"
      }
      elem: {
        name: "users"
      }
      elem: {
        name: "user"
        key: {
          key: "username"
          value: "test1"
        }
      }
      elem: {
        name: "config"
      }
      elem: {
        name: "password"
      }
    }
    val: {
      string_val: "$ABC123"
    }
  }
  update: {
    path: {
      elem: {
        name: "system"
      }
      elem: {
        name: "aaa"
      }
      elem: {

```

```

        name: "authentication"
      }
      elem: {
        name: "users"
      }
      elem: {
        name: "user"
        key: {
          key: "username"
          value: "test1"
        }
      }
    }
    elem: {
      name: "config"
    }
    elem: {
      name: "role"
    }
  }
  val: {
    string_val: "superuser"
  }
}
}

```

```

update: {
  timestamp: 1676295464
  update: {
    path: {
      elem: {
        name: "system"
      }
      elem: {
        name: "aaa"
      }
      elem: {
        name: "authentication"
      }
      elem: {
        name: "users"
      }
    }
  }
}

```



```

    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test1"
      }
    }
  }
  elem: {
    name: "config"
  }
  elem: {
    name: "username"
  }
}
val: {
  string_val: "test1"
}
}
update: {
  path: {
    elem: {
      name: "system"
    }
    elem: {
      name: "aaa"
    }
    elem: {
      name: "authentication"
    }
    elem: {
      name: "users"
    }
    elem: {
      name: "user"
      key: {
        key: "username"
        value: "test1"
      }
    }
  }
  elem: {
    name: "config"
  }
  elem: {

```

```
        name: "password"
      }
    }
    val: {
      string_val: "$ABC123"
    }
  }
  update: {
    path: {
      elem: {
        name: "system"
      }
      elem: {
        name: "aaa"
      }
      elem: {
        name: "authentication"
      }
      elem: {
        name: "users"
      }
      elem: {
        name: "user"
        key: {
          key: "username"
          value: "test1"
        }
      }
    }
    elem: {
      name: "config"
    }
    elem: {
      name: "role"
    }
  }
  val: {
    string_val: "superuser"
  }
}
```

RELATED DOCUMENTATION

| [gRPC Services for Junos Telemetry Interface](#)

Configure gRPC Services

SUMMARY

Configure the gRPC server to enable a client to use gRPC services on the network device, including: gRPC Network Operations Interface (gNOI) services, gRPC Network Management Interface (gNMI) services, and gRPC Routing Information Base Interface (gRIBI) services.

IN THIS SECTION

- [Understanding Authentication and Authorization for gRPC-Based Services | 52](#)
- [Obtain X.509 Certificates | 54](#)
- [Load the gRPC Server's Local Certificate in the Junos PKI | 57](#)
- [Enable gRPC Services | 58](#)
- [Configure Mutual \(Bidirectional\) Authentication for gRPC Services | 63](#)
- [Configure the User Account for gRPC Services | 68](#)
- [Configure gRPC RPC Authorization | 69](#)

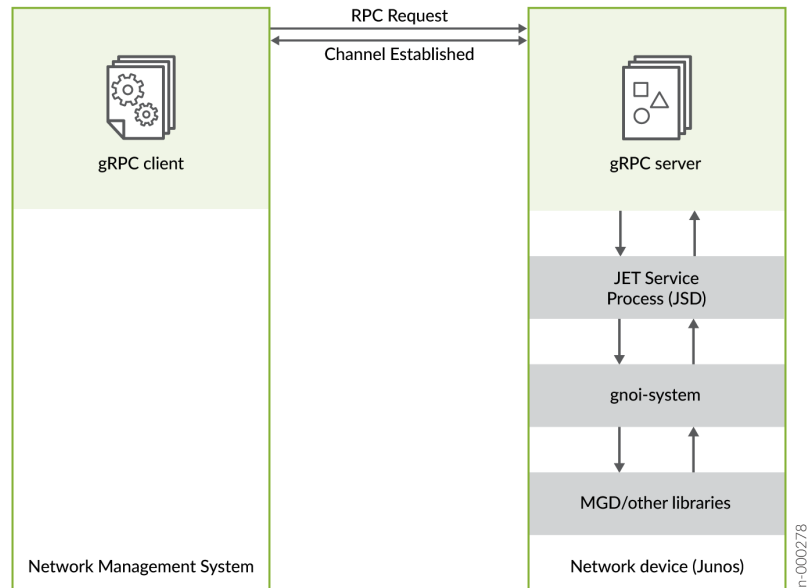
This topic discusses how to configure gRPC services on Junos devices, including the options for authentication and how to configure each option. Before the server and client can establish a gRPC session, you must satisfy the requirements discussed in the following sections:

- ["Understanding Authentication and Authorization for gRPC-Based Services" on page 52](#)
- ["Obtain X.509 Certificates" on page 54](#)
- ["Load the gRPC Server's Local Certificate in the Junos PKI" on page 57](#)
- ["Enable gRPC Services" on page 58](#)
- ["Configure Mutual \(Bidirectional\) Authentication for gRPC Services" on page 63 \(Optional\)](#)
- ["Configure the User Account for gRPC Services" on page 68](#)
- ["Configure gRPC RPC Authorization" on page 69 \(Optional\)](#)

Understanding Authentication and Authorization for gRPC-Based Services

The gNMI, gNOI, and gRIBI interfaces use the gRPC Remote Procedure Call framework for transport. The gRPC server runs on the network device and listens for connection requests on a specified port. The gRPC client application runs on a remote network management system (NMS) and establishes a gRPC channel with the server on the specified host and port. The client executes RPCs through the SSL-encrypted gRPC session to perform network service operations. [Figure 3 on page 52](#) illustrates a simple connection between a gRPC client and server.

Figure 3: gRPC Server and Client Interaction



gRPC channels use channel credentials to handle authentication between the server and the client. Standard channel credentials use X.509 digital certificates for authenticating the server and the client. A digital certificate provides a way of authenticating users through a trusted third-party called a *certificate authority* or *certification authority (CA)*. The CA verifies the identity of a certificate holder and “signs” the certificate to attest that it has not been forged or altered. The X.509 standard defines the format for the certificate. Digital certificates can be used to establish a secure connection between two endpoints through certificate validation. To establish a gRPC channel, each endpoint (device or application) that requires authentication must supply an X.509 certificate in the exchange.

Junos devices support both server-only authentication as well as mutual authentication for SSL and TLS-based gRPC sessions. When server-only authentication is configured, the server provides its public key certificate when the channel is established. The client uses the server's Root CA certificate to

authenticate the server. When mutual authentication is configured, the client also provides its certificate when it connects to the server, and the server validates the certificate. If the certificate validation is successful, the client is allowed to make calls. We recommend that you configure mutual authentication and use CA-signed certificates for the strongest security, although self-signed certificates are accepted.

A public key infrastructure (PKI) supports the distribution and identification of public encryption keys, enabling users to both securely exchange data over networks such as the Internet and verify the identity of the other party. For gRPC-based services, the Junos PKI must contain the certificate for the local device acting as the gRPC server. If you use mutual authentication, the Junos PKI must also contain the Root CA certificates required to validate the certificates of any gRPC clients that connect to the device.

[Table 6 on page 53](#) outlines the general requirements for server-only authentication and mutual authentication when a gRPC client connects to the device to perform gRPC-based services. The gRPC server's certificate must define either the server's hostname in the Common Name (CN) field, or it must define the server's IP address in the Subject Alternative Name (subjectAltName or SAN) IP Address field. The client application must use the same value to establish the connection to the server. If the certificate defines the SubjectAltName IP Address field, the Common Name field is ignored during authentication.

Table 6: Requirements for Server-Only and Mutual Authentication for gRPC Sessions

Requirements	Server-Only Authentication	Mutual Authentication
Certificates	<p>The server must have an X.509 public key certificate.</p> <p>If the client connects to the server's IP address instead of the hostname, the server's certificate must include the subjectAltName (SAN) IP address extension field with the IP address of the server.</p>	<p>The server and client must each have an X.509 public key certificate.</p> <p>If the client connects to the server's IP address instead of the hostname, the server's certificate must include the subjectAltName (SAN) IP address extension field with the IP address of the server.</p>
Junos PKI	The server's local certificate must be loaded in the Junos PKI.	The server's local certificate and each client's Root CA certificate must be loaded in the Junos PKI.
Channel credentials	The client must pass in the server's Root CA certificate when the gRPC channel is established.	The client must pass in their certificate and key and the server's Root CA certificate when the gRPC channel is established.

Channel credentials are attached to the gRPC channel and enable the client application to access the service. Call credentials, on the other hand, are attached to a specific service operation (RPC request) and provide information about the person who is using the client application. Call credentials are sent

per request, that is, for each RPC call. To execute gRPC-based operations on Junos devices, you must provide call credentials in the request. The user must either have a user account defined locally on the device, or the user must be authenticated by a TACACS+ server, which then maps the user to a user template account that is defined locally on the device. You can provide the call credentials (username and password) in the RPC's `metadata` argument. If authentication is successful, the Junos device executes the RPC request using the account privileges of the specified user.



NOTE: As an alternative to passing in call credentials for every RPC executed on a Junos device, you can use the Juniper Extension Toolkit `jnx_authentication_service` API to log in to the device once at the start of the gRPC session, and all subsequent RPCs executed in the channel are authenticated. You can download the JET Client IDL library from the [Juniper Networks download site](#).

By default, Junos devices authorize an authenticated gRPC client to execute all gRPC RPCs. You can optionally configure a gRPC user's login class to explicitly allow or deny specific gRPC RPCs. To specify the RPCs, you configure the `allow-grpc-rpc-regexps` and `deny-grpc-rpc-regexps` statements and define regular expressions that match the RPCs. See ["Configure gRPC RPC Authorization" on page 69](#) for more information.

Obtain X.509 Certificates

A gRPC session uses X.509 public key certificates to authenticate the gRPC server and client. For server-only authentication, the gRPC server must have a certificate. For mutual authentication, both the gRPC server and client must have certificates. The requirements for the certificates are:

- The certificate can be signed by a CA or self-signed.
- The certificate must be PEM-encoded.
- The gRPC server's certificate must define either the gRPC server's hostname in the Common Name (CN) field, or it must define the gRPC server's IP address in the SubjectAltName (SAN) IP Address field. The gRPC client must use the same value to establish the connection to the server. If the certificate defines the SubjectAltName IP Address, the Common Name field is ignored during authentication.

To use OpenSSL to obtain the gRPC server's certificate:

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

```
user@nms:~$ openssl genrsa -out server.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
```

```
...++++
.....++++
e is 65537 (0x010001)
```

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

```
user@nms:~$ cat openssl.cnf
# OpenSSL configuration file.
...
extensions          = v3_sign
...
[v3_sign]
subjectAltName=IP:10.53.52.169
```

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

```
user@nms:~$ openssl req -new -key server.key -out server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:Sunnyvale
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Acme
Organizational Unit Name (eg, section) []: testing
Common Name (e.g. server FQDN or YOUR name) []:gnoi-server.example.com
Email Address []:
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Alternatively, you can provide the CSR information in a single command, for example:

```
user@nms:~$ openssl req -new -key server.key -out server.csr -subj "/C=US/ST=CA/L=Sunnyvale/
O=Acme/OU=testing/CN=gnoi-server.example.com"
```

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

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

```
user@nms:~$ openssl x509 -req -in server.csr -CA /etc/pki/certs/ServerRootCA.crt -
CAkey /etc/pki/certs/ServerRootCA.key -set_serial 0101 -out server.crt -days 365 -
extfile openssl.cnf
Signature ok
subject=C = US, ST = CA, L = Sunnyvale, O = Acme, OU = testing, CN = gnoi-
server.example.com
Getting Private key
```

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

```
user@nms:~$ openssl x509 -req -in server.csr -signkey server.key -out server.crt -days
365 -sha256 -extfile openssl.cnf
Signature ok
subject=C = US, ST = CA, L = Sunnyvale, O = Acme, OU = testing, CN = gnoi-
server.example.com
Getting Private key
```

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

```
user@nms:~$ openssl x509 -text -noout -in server.crt
Certificate:
  Data:
    Version: 3 (0x2)
    ...
    Subject: C = US, ST = CA, L = Sunnyvale, O = Acme, OU = testing, CN = gnoi-
server.example.com
    ...
```



```
X509v3 extensions:
    X509v3 Subject Alternative Name:
        IP Address:10.53.52.169
    ...
```

For mutual authentication, repeat the previous steps with the information for the gRPC client to generate the client's key and certificate. The client certificate does not require the SAN IP extension field.

Load the gRPC Server's Local Certificate in the Junos PKI

The network device running the gRPC server must have an X.509 certificate that identifies the device to gRPC clients. To perform gRPC-based services on the Junos device, you must load the public key certificate and key for the local network device in the Junos PKI. After you load the certificate and perform the initial configuration, gRPC clients can then use any microservice to update the certificate. For example, a gRPC client can use the gNOI CertificateManagement service to install a new certificate or replace an existing certificate.

To load the local device's certificate and key in the PKI:

1. Download the certificate and key for the device that is acting as the gRPC server to that device.
2. In operational mode, define an identifier and load the local device's certificate and key into the PKI.

```
user@host> request security pki local-certificate load certificate-id certificate-id filename
path-to-certificate-file key path-to-key-file
Local certificate loaded successfully
```

For example:

```
user@host> request security pki local-certificate load certificate-id gnoi-server
filename /var/tmp/server.crt key /var/tmp/server.key
Local certificate loaded successfully
```

3. (Optional) Verify the certificate is present in the PKI database.

```
user@host> show security pki local-certificate certificate-id gnoi-server
LSYS: root-logical-system
Certificate identifier: gnoi-server
Issued to: gnoi-server.example.com, Issued by: C = US, ST = CA, O = serverRootCAOrg, CN =
```

```
serverRootCA
  Validity:
    Not before: 04-13-2022 18:15 UTC
    Not after: 04-13-2023 18:15 UTC
  Public key algorithm: rsaEncryption(4096 bits)
  Keypair Location: Keypair generated locally
```

Enable gRPC Services

IN THIS SECTION

- [\[edit system services http servers\]](#) | 59
- [\[edit system services extension-service request-response grpc ssl\]](#) | 61

gRPC-based services use an API connection setting based on Secure Socket Layer (SSL) or Transport Layer Security (TLS) technology. For these connections, you must specify a local certificate that identifies the gRPC server.

After you enable gRPC services and specify a local certificate, the network device uses server-only authentication. You can then optionally configure mutual authentication by completing the steps described in "[Configure Mutual \(Bidirectional\) Authentication for gRPC Services](#)" on page 63.

You can configure your network device for gRPC services and specify the local certificate used for server authentication at one of the following hierarchy levels:

- [\[edit system services http servers\]](#)—Use this statement hierarchy to configure one or more gRPC servers that host different sets of services on unique ports. Additionally, each server can support different listening addresses, certificates, and routing instances.
- [\[edit system services extension-service request-response grpc ssl\]](#)—Use this statement hierarchy when you require only a single gRPC server that supports all gRPC services on the same listening address and port.

To configure the device for gRPC services, follow the instructions for the hierarchy level that meets your environment's requirements.

[edit system services http servers]

To configure one or more gRPC servers at the [edit system services http servers] hierarchy level:

1. Navigate to the gRPC servers hierarchy level and specify an identifier for the server.

```
[edit]
user@host# edit system services http servers server name
```

For example:

```
[edit]
user@host# edit system services http servers server grpc-server1
```

2. Configure the port to use for the gRPC services. The port must be unique for each gRPC server.

```
[edit system services http servers server name]
user@host# set port port-number
```

For example:

```
[edit system services http servers server grpc-server1]
user@host# set port 32767
```

3. Configure the gRPC services hosted by this server.

```
[edit system services http servers server name]
user@host# set grpc [service1 service2 ...]
```

In this example, the server hosts gNMI services and gNOI services.

```
[edit system services http servers server grpc-server1]
user@host# set grpc [gnmi gnoi]
```

4. Specify the local certificate that identifies the server to a client.

Enter the identifier for the local certificate that you previously loaded into the Junos PKI with the `request security pki local-certificate load operational mode` command.

```
[edit system services http servers server name]  
user@host# set tls local-certificate certificate-id
```

The following example configures the local certificate `gnoi-server`:

```
[edit system services http servers server grpc-server1]  
user@host# set tls local-certificate gnoi-server
```

5. (Optional) Specify the IPv4 or IPv6 address on which the server listens for incoming connections.

```
[edit system services http servers server name]  
user@host# set listen-address address
```

For example:

```
[edit system services http servers server grpc-server1]  
user@host# set ip-address 192.168.2.1
```



NOTE: If you do not specify an IP address, the default address of `::` is used to listen for incoming connections.

6. (Optional) Configure the routing-instance to use for this gRPC server, if different from the default routing instance.

```
[edit system services http servers server name]  
user@host# set routing-instance routing-instance
```

The following example uses the `mgmt-junos` routing instance.

```
[edit system services http servers server grpc-server1]  
user@host# set routing-instance mgmt-junos
```

7. (Optional) Configure the maximum number of connections that this gRPC server supports.

```
[edit system services http servers server name]
user@host# set max-connections connections
```

The following example configures a maximum of 10 connections. The default is 5.

```
[edit system services http servers server grpc-server1]
user@host# set max-connections 10
```

8. Commit the configuration.

```
user@host# commit
```

To configure mutual authentication instead of server-only authentication, you must also complete the steps in ["Configure Mutual \(Bidirectional\) Authentication for gRPC Services" on page 63](#).

[edit system services extension-service request-response grpc ssl]

To configure a single gRPC server at the [edit system services extension-service request-response grpc ssl] hierarchy level:

1. Navigate to the SSL-based API connection settings for gRPC services.

```
[edit]
user@host# edit system services extension-service request-response grpc ssl
```

2. Configure the port to use for gRPC services.

```
[edit system services extension-service request-response grpc ssl]
user@host# set port port-number
```

For example:

```
[edit system services extension-service request-response grpc ssl]
user@host# set port 32767
```

3. Specify the local certificate that identifies the server to a client.

Enter the identifier for the local certificate that you previously loaded into the Junos PKI with the `request security pki local-certificate load operational mode` command.

```
[edit system services extension-service request-response grpc ssl]  
user@host# set local-certificate certificate-id
```

The following example configures the local certificate `gnoi-server`:

```
[edit system services extension-service request-response grpc ssl]  
user@host# set local-certificate gnoi-server
```

4. Configure the device to use the PKI database for certificates.

```
[edit system services extension-service request-response grpc ssl]  
user@host# set use-pki
```

5. Enable the device to reload certificates without terminating the gRPC session.

```
[edit system services extension-service request-response grpc ssl]  
user@host# set hot-reloading
```

6. (Optional) Specify an IP address to listen to for incoming connections.

```
[edit system services extension-service request-response grpc ssl]  
user@host# set ip-address address
```

For example:

```
[edit system services extension-service request-response grpc ssl]  
user@host# set ip-address 192.168.2.1
```



NOTE: If you do not specify an IP address, the default address of `::` is used to listen for incoming connections.

7. (Optional) Configure tracing for extension services to debug any issues that might arise.

```
[edit]
user@host# top
user@host# set system services extension-service traceoptions file jsd
user@host# set system services extension-service traceoptions flag all
```



NOTE: To view Junos OS Evolved trace files for extensions services, use the `show trace application jsd` and `show trace application jsd live` operational mode commands.

8. Commit the configuration.

```
user@host# commit
```

To configure mutual authentication instead of server-only authentication, you must also complete the steps in ["Configure Mutual \(Bidirectional\) Authentication for gRPC Services" on page 63](#).

Configure Mutual (Bidirectional) Authentication for gRPC Services

IN THIS SECTION

- [Configure Mutual Authentication in the Device Configuration | 64](#)
- [Configure Mutual Authentication Using the gNOI CertificateManagement Service | 67](#)

You can configure mutual (bidirectional) authentication for gRPC sessions, which authenticates both the network device as the gRPC server and the NMS as the gRPC client using certificates. The Junos device uses the credentials provided by the external client to authenticate the client and authorize a connection.

You can configure mutual authentication on Junos devices using one of the following options:

- Configure the mutual authentication settings directly in the configuration.
- Set up server-only authentication initially, and then use the gNOI CertificateManagement service to load the necessary CA certificates on the device.

If you configure mutual authentication directly in the device configuration, the device configuration takes precedence over any setup done using the gNOI services.

Before you begin:

- Load the certificate and key for the network device acting as the gRPC server into the device's PKI as described in ["Load the gRPC Server's Local Certificate in the Junos PKI" on page 57](#).
- Enable gRPC services and configure the local server authentication as described in ["Enable gRPC Services" on page 58](#).

The following sections discuss the different methods for configuring mutual authentication. You can use whichever method works best for your environment.

Configure Mutual Authentication in the Device Configuration

To configure authentication for the gRPC client directly in the network device configuration:

1. Download the root CA certificate that will be used to validate the client's certificate to the local device acting as the gRPC server.
2. Configure a CA profile for the client certificate's root CA at the `[edit security pki]` hierarchy.

```
[edit security pki]
user@host# set ca-profile ca-profile-name ca-identity ca-identifier
```

For example:

```
[edit security pki]
user@host# set ca-profile gnoi-client ca-identity clientRootCA
```

3. Commit the configuration.

```
[edit]
user@host# commit and-quit
```

4. In operational mode, load the root CA certificate that will be used to verify the client's certificate into the Junos PKI. Specify the `ca-profile` identifier that you configured in the previous steps.

```
user@host> request security pki ca-certificate load ca-profile ca-profile filename cert-path
```


For example:

```
user@host> request security pki ca-certificate load ca-profile gnoi-client filename /var/tmp/
clientRootCA.crt
Fingerprint:
  00:2a:30:e9:59:94:db:f1:a1:5c:d1:c9:d4:5f:db:8f:f1:f0:8d:c4 (sha1)
  02:3b:a0:b8:95:0c:a2:fa:15:18:57:3d:a3:10:e9:ac (md5)

69:97:90:39:de:75:a0:1d:94:1e:06:a8:be:8c:66:e5:41:95:fd:dc:14:8a:e7:3a:e0:42:9e:f9:f7:dd:c8:c
2 (sha256)
Do you want to load this CA certificate ? [yes,no] (no) yes

CA certificate for profile gnoi-client loaded successfully
```



TIP: To load a CA certificate bundle, issue the `request security pki ca-certificate ca-profile-group load ca-group-name ca-group-name filename bundle-path` command.

After loading the certificate, enter configuration mode and continue configuring mutual authentication. You must configure mutual authentication under the same hierarchy level where you configured your server. Perform the steps outlined in the section for your hierarchy level.

[edit system services http servers]

To configure mutual authentication for a server configured at the [edit system services http servers] hierarchy level:

1. Navigate to the `tls` statement under your server configuration.

```
[edit]
user@host# edit system services http servers server name tls
```

For example:

```
[edit]
user@host# edit system services http servers server grpc-server1 tls
```

2. Enable mutual authentication and specify the requirements for client certificates.

```
[edit system services http servers server name tls]
user@host# set mutual-authentication authentication-type requirement
```

For example, to specify the strongest authentication, which requires a certificate and its validation, use `request-and-require-cert-and-verify`, which is also the default.

```
[edit system services http servers server grpc-server1 tls]
user@host# set mutual-authentication authentication-type request-and-require-cert-and-verify
```

3. Specify the CA profile that will be used to verify the client certificate.

The CA profile was configured in step "2" on page 64 of the CA profile configuration.

```
[edit system services http servers server name tls]
user@host# set mutual-authentication certificate-authority certificate-authority
```

For example, to specify the CA profile named `gnoi-client`:

```
[edit system services http servers server grpc-server1 tls]
user@host# set mutual-authentication certificate-authority gnoi-client
```

4. Commit the configuration.

```
[edit system services http servers server name tls]
user@host# commit and-quit
```

[edit system services extension-service request-response grpc ssl]

To configure mutual authentication for a server configured at the `[edit system services extension-service request-response grpc ssl]` hierarchy level:

1. Enable mutual authentication and specify the requirements for client certificates.

```
[edit system services extension-service request-response grpc ssl]
user@host# set mutual-authentication client-certificate-request requirement
```

For example, to specify the strongest authentication, which requires a certificate and its validation, use `require-certificate-and-verify`.

```
[edit system services extension-service request-response grpc ssl]
user@host# set mutual-authentication client-certificate-request require-certificate-and-verify
```



NOTE: The default is `no-certificate`. The other options are: `request-certificate`, `request-certificate-and-verify`, `require-certificate`, `require-certificate-and-verify`.

We recommend that you use the `no-certificate` option in a test environment only.

2. Specify the CA profile that will be used to verify the client certificate.

The CA profile was configured in step "2" on page 64 of the CA profile configuration.

```
[edit system services extension-service request-response grpc ssl]
user@host# set mutual-authentication certificate-authority certificate-authority
```

For example, to specify the CA profile named `gnoi-client`:

```
[edit system services extension-service request-response grpc ssl]
user@host# set mutual-authentication certificate-authority gnoi-client
```

3. Commit the configuration.

```
[edit system services extension-service request-response grpc ssl]
user@host# commit and-quit
```

Configure Mutual Authentication Using the gNOI CertificateManagement Service

You can use the gNOI CertificateManagement service to set up mutual authentication between the gRPC client and gRPC server instead of configuring the settings directly in the device configuration. You initially set up server-only authentication and then use the gNOI CertificateManagement service RPCs to load the client CA certificates. See *gNOI Certificate Management Service* for information about loading the certificates using the gNOI CertificateManagement service.

The gRPC server supports only one global CA certificate bundle for gNOI services. When you use the gNOI CertificateManagement service to load the CA certificate bundle, the device implicitly uses mutual authentication. However, you should take note of the following:

- The CertificateManagement service always loads the CA certificate bundle using the ca-profile-group reserved identifier gnoi-ca-bundle.
- If you use the CertificateManagement service to load the CA certificate bundle, the device implicitly uses mutual authentication.
- If the CertificateManagement service sends a request to load a new CA certificate bundle, the server clears the certificates for the previous CA bundle from the device and loads the new ones.
- If you use the CertificateManagement service to load a CA certificate bundle and you also explicitly configure mutual authentication in the device configuration, then the configured statements take precedence.

Configure the User Account for gRPC Services

Channel credentials are attached to the gRPC channel and enable the client application to access the service. Call credentials are attached to a specific RPC request and provide information about the user who is using the client application. You must provide call credentials in each RPC request, which requires a user account for the network device. The user must have a user account defined locally on the network device, or the user must be authenticated by a TACACS+ server, which then maps the user to a user template account that is defined locally on the device.

To create a user account:

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

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

2. For local user accounts, configure the user's password.

You can omit the password for local user template accounts because the user is authenticated through a remote authentication server.

```
[edit system login]
user@host# set user gnoi-user authentication plain-text-password
New password:
Retype new password:
```

3. (Optional) Configure the `full-name` statement to specify the user's name.

```
[edit system login]
user@host# set user gnoi-user full-name "gNOI client"
```

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

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

5. Repeat the preceding steps on each network device where the gRPC client will execute RPCs in a gRPC session.

Configure gRPC RPC Authorization

By default, Junos devices authorize an authenticated gRPC client to execute all gRPC RPCs. You can configure a Junos login class to explicitly allow or deny gRPC RPCs. To specify the RPCs, you configure the `allow-grpc-rpc-regexps` and `deny-grpc-rpc-regexps` statements and define regular expressions that match the RPCs. If there are conflicting expressions in the allow and deny lists, the deny list takes precedence. If an RPC does not match either list, the RPC is allowed by default.

Junos devices use the following syntax for specifying gRPC RPCs:

```
/package.service/rpc
```

Where *package*, *service*, and *rpc* are the names defined in the respective statement in that service's proto definition file. For example:

```
/gnmi.gNMI/Get
/gnoi.certificate.CertificateManagement/Rotate
/gnoi.system.System/Reboot
/gnoi.system.System/RebootStatus
/gribi.gRIBI/*
```

You can configure multiple `allow-grpc-rpc-regexps` and `deny-grpc-rpc-regexps` statements with one or more expressions. Enclose each expression within quotation marks (" "). Enclose multiple expressions in square brackets [], and separate the expressions with a space.

```
allow-grpc-rpc-regexps ["regex1" "regex2" ... ]
allow-grpc-rpc-regexps "regex3"
```

To create a login class that defines authorization for gRPC RPCs:

1. Configure the login class name and permissions.

```
[edit system login]
user@host# set class class-name permissions [permission1 permission2 ...]
```

For example:

```
[edit system login]
user@host# set class grpc-operator permissions all
```

2. Within the class, configure regular expressions for the RPCs that the class allows.

```
[edit system login class class-name]
user@host# set allow-grpc-rpc-regexps ["regex1" "regex2" ... ]
user@host# set allow-grpc-rpc-regexps "regex3"
```

For example, the following statement allows the gNMI Get() RPC and all gNOI System service RPCs.

```
[edit system login class grpc-operator]
user@host# set allow-grpc-rpc-regexps ["/gnmi.gNMI/Get" "/gnoi.system.System/.*"]
```

3. Configure regular expressions for the RPCs that the class denies.

```
[edit system login class class-name]
user@host# set deny-grpc-rpc-regexps ["regex1" "regex2" ... ]
```

For example, the following statements deny the gNMI Set() RPC and also deny all RPCs for the gRIBI service as well as the gNOI CertificateManagement service.

```
[edit system login class grpc-operator]
user@host# set deny-grpc-rpc-regexps ["/gnmi.gNMI/Set" "/gribi.gRIBI/.*"]
user@host# set deny-grpc-rpc-regexps "/gnoi.certificate.CertificateManagement/.*"

```

4. Assign the login class to the appropriate gRPC users.

```
[edit system login]
user@host# set user username class class-name

```

For example, the following statement assigns the grpc-operator class to the grpc-user user.

```
[edit system login]
user@host# set user grpc-user class grpc-operator

```

After enabling gRPC services on the network device, set up the remote NMS as a gRPC client. To enable the client to execute gNOI operations, configure the client as outlined in *Configure gNOI Services*.

Change History Table

Feature support is determined by the platform and release you are using. Use [Feature Explorer](#) to determine if a feature is supported on your platform.

Release	Description
25.2R1 & 25.2R1-EVO	Starting in Junos OS Release 25.2R1 and Junos OS Evolved Release 25.2R1, you can configure multiple gRPC servers that host different sets of services on unique ports.

Establish a Dial-in Telemetry Connection

SUMMARY

This topic describes the procedure to establish a dial-in telemetry connection. Configure the gRPC service and data collector, and select sensor paths based on

IN THIS SECTION

- [Configure the Data Collector | 72](#)

the sensor information you want to collect from the Junos device.

- [Selecting Sensor Paths | 73](#)
- [Decoding Data on the Collector | 74](#)

In dial-in mode, the data collector initiates the connection to the network device and subscribes to telemetry data. A session is established between the collector and the device, and then data is streamed from the device to the collector (streaming intervals can be configured). This method is preferred by operators who want a single configuration and operational data channel.

Prerequisites:

1. Ensure the Juniper device runs a compatible Junos OS version that supports JTI.
2. Set up a gRPC-compatible telemetry collector (for example, Juniper Telemetry Collector, Prometheus, or InfluxDB) to receive and process telemetry data.
3. Ensure the Juniper device and collector can communicate over the network (TCP port 50051 for gRPC (default), or a custom port).
4. If your implementation uses TLS, prepare certificates for the device and collector.
5. For a complete list of sensors supported for your Juniper device, see [Junos YANG Data Model Explorer](#).
 - **Note:** The [Junos YANG Data Model Explorer](#) was introduced in the 23.2R2-S2 releases. From releases 20.2R1 up to 23.1R1, the sensor information is available in the [Junos Telemetry Sensor Explorer](#).
6. Configure gRPC services, see "[Configure gRPC Services](#)" on page 51.

Follow the procedures listed below to establish a dial-in telemetry connection:

Configure the Data Collector

1. The collector must be ready to initiate the connection. Configure the gRPC client to connect to the Juniper device, where:
 - IP Address: The device's management IP (192.168.1.1).
 - Port: 50051 (or custom port).
 - Protocol: gRPC with gNMI or JET subscription.
 - Credentials: If TLS is enabled, provide the client certificate and key.

- Subscription: Specify the sensor paths (for example, /junos/system/linecard/interface/) and subscription mode (for example, stream).

Install and configure the gNMI client on the collector. Ensure the protobuf definitions are available on the collector. This information is required to decode the sensor data received at the collector. The gNMI client is available on Git Hub, see [gNMI client on Github](#).

Run the following command:

```
gnmic -a <device_ip>:<port> subscribe --path <path> --mode stream
```

For example:

```
gnmic -a 192.168.1.1:50051 subscribe --path /junos/system/linecard/interface/ --mode stream
```

2. To verify the configuration, you can use show agent sensors and show extension-service request-response clients commands.
3. On the data collector, verify if the data is received. For example, if Prometheus or Grafana is used as a data collector, check the logs or visualizations.

Selecting Sensor Paths

IN THIS SECTION

- [Sensor Explorer and Guidelines for Selecting Sensor Paths](#) | 73

Sensor Explorer and Guidelines for Selecting Sensor Paths

Use the Juniper Networks [Junos YANG Data Model Explorer](#) to view all the supported resource paths, their corresponding leaves, and the device platforms that support them.



NOTE: The [Junos YANG Data Model Explorer](#) was introduced in the 23.2R2-S2 releases. From releases 20.2R1 up to 23.1R1, the sensor information is available in the [Junos Telemetry Sensor Explorer](#).

To search and view other telemetry sensors and specific information about some legacy sensors, see ["Legacy Sensor Paths" on page 255](#).

For information on guidelines and best practices for configuring sensor paths, see ["Explore Sensor Paths" on page 9](#).

Decoding Data on the Collector

IN THIS SECTION

- [Decode data on the collector | 74](#)

Decode 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 < ../data.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 . < data.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"
    }
}
}
```

3

PART

Dial-out Telemetry

- Understanding Dial-Out Telemetry | **88**
 - Guidelines for Streaming Telemetry Data Over UDP | **89**
 - Configure gRPC service for Legacy Dial-Out Connection | **91**
 - Establish a Dial-out Telemetry Connection | **94**
-

Understanding Dial-Out Telemetry

SUMMARY

Dial-out telemetry is a method used in network monitoring where the device (like a router) initiates the connection to send data to a collector. In dial-out telemetry, the device "dials out" to the collector, meaning it sends the initial SYN packet to establish the connection. This approach simplifies network management because it avoids the need to open ports for inbound management traffic.

IN THIS SECTION

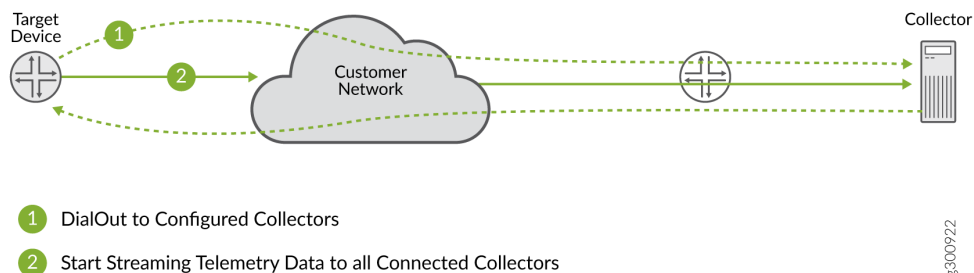
- [Benefits of Using Dial-Out Telemetry | 89](#)

Starting with Junos OS Release 22.4R1, JTI supports remote gRPC dial-out support on ACX Series routers, MX Series routers, PTX Series routers, and QFX Series switches. With gRPC dialout, the target device (server) initiates a gRPC session with the collector (client). When the session is established, the target streams the telemetry data that is specified by the sensor-group subscription to the collector. This is in contrast to the gRPC network management interface (gNMI) dial-in method, in which the collector initiates a connection to the target device.

gRPC dial-out simplifies streaming telemetry statistics. Configuring the target device to stream statistics and export them to a collector IP address removes the burden of access being placed on the collector (client).

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.

Figure 4: gRPC Dial-Out



Benefits of Using Dial-Out Telemetry

- Reduces target device exposure to threats outside of their topology.
- Simplifies access to a target device. The gRPC Dial-In method requires a collector to overcome a series of complex firewall configurations to gain access to the target device. gRPC Dial-Out does not.
- Collectors can be stateless; without the need to initiate a session, they simply listen, subscribe, and store collected data.
- Support mutual encryption for heightened security.

Guidelines for Streaming Telemetry Data Over UDP

SUMMARY

This section describes guidelines for streaming telemetry data over a UDP connection.

Streaming Telemetry Data over UDP is based on the dial-out mechanism. The sensor paths are configured through the CLI, and the device sends the data for the configured sensor paths over UDP to the collector's destination address. The destination address is configured through the CLI. While streaming telemetry over UDP from native sensors, the data is sent to the collector over UDP in the *protobuf* format. For *protobuf* format information see, ["Native Protocol Buffers \(protobuf\) Message Structure" on page 12](#).

The CLI configuration procedure to stream telemetry data over UDP includes configuring a sensor profile (where resource parameters are specified), a streaming server profile (where remote address and remote-port parameters are specified), and an export profile (where transport protocol, reporting rate, format, and payload-size parameters are specified). To configure a streaming UDP connection, see ["Establish a Dial-out Telemetry Connection" on page 94](#).

To decode Junos Telemetry Interface UDP Data at the Collector, see [Decoding Junos Telemetry Interface Data With UNIX Utilities](#).



NOTE:

1. Telemetry data is streamed directly from the source application to the collector over UDP. Therefore, the configuration is determined by the type of connection to the collector. The collector must be reachable over the management interface or a WAN interface.
2. In the case of line card sensors, the line cards can directly export the data to the collector if the remote address is reachable over the WAN interface. If the telemetry data must be exported over the management interface for line card sensors, it must be routed to the Routing Engine and then sent to the collector. In this scenario, configure the following hidden knob:

```
set services analytics sensor resource_test export-to-routing-engine
```

3. Only the *.gpb* format is supported for transporting telemetry data over UDP. Below is a sample configuration to stream telemetry data over UDP for the sensor path `/junos/system/linecard/interface/`. Data is exported every “10” seconds to the collector with a maximum payload size of “5000” to the remote address 143.1.1.2 with port number 3026:

```
set services analytics streaming-server server_test remote-address 143.1.1.2
set services analytics streaming-server server_test remote-port 3026
set services analytics export-profile export_test local-port 0
set services analytics export-profile export_test reporting-rate 10
set services analytics export-profile export_test payload-size 5000
set services analytics export-profile export_test format gpb
set services analytics export-profile export_test transport udp
set services analytics sensor resource_test server-name server_test
set services analytics sensor resource_test export-name export_test
set services analytics sensor resource_test resource /junos/system/linecard/
interface/
```

Configure gRPC service for Legacy Dial-Out Connection

SUMMARY

In legacy gRPC dial-out connections, the outgoing interface IP address is used as the source address. From release 24.2 onwards, you can configure a source IP address and routing instance for gRPC dial-out connections. This section describes the procedures for configuring the source IP address and routing instance for gRPC dial-out connections.

IN THIS SECTION

- [Configure an IP Source Address for Legacy gRPC Dial-Out Connections | 91](#)
- [Configure a Routing Instance for Legacy gRPC Dial-Out Connections | 92](#)

Configure an IP Source Address for Legacy gRPC Dial-Out Connections

Starting with Junos OS Evolved Release 24.2R1, JTI supports configuring a source IP address for legacy gRPC service dial-out connections on ACX Series routers, PTX Series routers, and QFX Series switches. In earlier releases that support legacy gRPC dial-out, the outgoing interface IP address is used as the source address without an option to configure a source IP address. This feature supports FLEX Deployments, providing the ability to send dial-out from the router's specified IP address or interface address (such as a loopback0 address).

If you do not configure a local address, the default local address is used.

To configure a local address:

1. In configuration mode, go to the [edit services analytics export-profile] hierarchy level and add the name of your export profile (here **ep1**).

```
user@host# edit services analytics export-profile ep1
```

2. Include the local-address statement (here, with IPv6 address **2000:200::20**).

You can only use an IPv6 address for legacy gRPC dial-out connections (transport type `grpc`). IPv6 addresses are not supported by UDP transport. If UDP transport is configured, an error is displayed when you commit the configuration.

```
[edit services analytics export-profile ep1]
user@host# set local-address 2000:200::20
```

3. Use the following operational mode command to confirm the local address configuration.

```
user@host> show services
```

```
analytics {
  streaming-server remote-dialout-server {
    remote-address 10.220.23.206;
    remote-port 50051;
  }
  export-profile ep1 {
    local-address 2000:200::20;-- IPv6 address configured
    reporting-rate 10;
    format json-gnmi;
    transport grpc;
  }
}
```

The export file `ep1` shows the IPv6 address `2000:200::20` as the local address.

RELATED DOCUMENTATION

| *local-address (Junos Telemetry Interface)*

Configure a Routing Instance for Legacy gRPC Dial-Out Connections

Starting with Junos OS Evolved Release 24.2R1, JTI supports configuring a routing instance for legacy gRPC service dial-out connections on ACX Series routers, PTX Series routers, and QFX Series switches.

If you do not configure a routing instance, the default routing instance is used.

To configure a routing instance:

1. In configuration mode, go to the [edit services analytics export-profile] hierarchy level and add the name of your export profile (here **ep1**).

```
user@host# edit services analytics export-profile ep1
```

2. Include the routing-instance statement (here, with routing instance **mgmt-1**).

You can only configure a routing instance for legacy gRPC dial-out connections (transport type grpc. If UDP transport is configured, an error is displayed when you commit the configuration.

```
[edit services analytics export-profile ep1]
user@host# set routing-instance mgmt-1
```

3. Use the following operational mode command to confirm the routing instance configuration.

```
user@host> show services
```

```
show services
analytics {
  streaming-server remote-dialout-server {
    remote-address 10.220.23.206;
    remote-port 50051;
  }
  export-profile ep1 {
    local-address 2000:200::20;
    reporting-rate 10;
    format json-gnmi;
    transport grpc;
    routing-instance mgmt-1;
  }
}
```

The export file ep1 displays the routing instance as mgmt-1.

RELATED DOCUMENTATION

| *routing-instance (Junos Telemetry Interface)*

Establish a Dial-out Telemetry Connection

SUMMARY

This topic describes the procedure to establish a dial-out telemetry connection. Configure the gRPC service, streaming server profile, sensor profile, export profile, and data collector based on the sensor information you want to collect from the Junos device.

IN THIS SECTION

- [Configure a Streaming Server Profile | 95](#)
- [Configure a Sensor Profile | 96](#)
- [Configure an Export Profile | 98](#)
- [Configure the Data Collector | 103](#)
- [Verify the Junos Telemetry Interface Sensor Configuration | 103](#)
- [Selecting Sensor Paths | 105](#)
- [Decoding Junos Telemetry Interface Data With UNIX Utilities | 106](#)

In dial-out mode, the Juniper device initiates a connection to an external telemetry collector and streams telemetry data. This method is preferred when centralized management of telemetry data is required, as the device pushes data to the collector without requiring the collector to initiate connections.

- Ensure the Juniper device runs a compatible Junos OS version that supports JTI dial-out telemetry.
- Set up a telemetry collector (for example, Juniper Telemetry Collector, Prometheus, or InfluxDB) compatible with the chosen transport protocol (for example, UDP, gRPC).
- Ensure network connectivity between the Juniper device and the collector (for example, TCP port 50051 for gRPC or a custom port for UDP).
- If using TLS with gRPC, prepare certificates for the device and collector.
- Refer to the [Junos YANG Data Model Explorer](#) (introduced in Junos OS Release 23.2R2-S2) or [Junos Telemetry Sensor Explorer](#) (for releases 20.2R1 to 23.1R1) for supported sensors.
- For legacy gRPC dial-out connections you can configure the IP address and routing instance, see ["Configure gRPC service for Legacy Dial-Out Connection" on page 91](#).

To monitor a specific system resource, you configure a sensor. Each sensor configuration requires three main components:

- Streaming server profile—Specifies the server for collecting data and related parameters, including the destination IP address and port number. Before you begin configure a connection from your Juniper Networks device to a server that is using in-band management interfaces.
- 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.



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.

To enable export of statistics, include the `export-profile` and `sensor` statements at the `[edit services analytics]` hierarchy level. The export profile must include the reporting rate, the transport service (for example, gRPC), and the format (for example, gbp-gnmi). The sensor configuration should include the name of the collector (the server's name), the name of the export profile, and the resource path. An example of a resource path is `/interfaces/interface[name='fxp0']`.



NOTE: When configuring an export profile for gNMI dial out, the export profile parameters, such as `{'dscp', 'forwarding-class', 'payload-size'}` are not applicable. Configuring any of these options generates an error.

Follow the procedures listed below to establish a dial-out telemetry connection:

Configure 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
```

Configure 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-*
```

- 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
```

- 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:

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

Configure an Export Profile

IN THIS SECTION

- Platform-Specific Export Profile Behavior | 99

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.

Platform-Specific Export Profile Behavior

Use [Feature Explorer](#) to confirm platform and release support for specific features.

Use the following table to review platform-specific behaviors for your platforms:

Table 7: Platform-Specific Export Profile Behavior

Platform	Difference
MX Series	Starting with Junos OS Release 17.3R1 on MX Series routers only, you can specify a packet loss priority for an export profile. As a result, you can apply the appropriate packet loss priority to each sensor. Loss priority settings help determine which packets are dropped from the network during periods of congestion. Previously, you could specify only the forwarding class and the DSCP value in an export profile. The following packet loss priority settings are supported: high, low, medium-high and medium-low. For more information about packet loss priority settings, see Mapping PLP to RED Drop Profiles .

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
```



NOTE: Configuring the transport protocol as
udp
and the exported data format as
gpb-gnmi
together is erroneous.

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



NOTE: The default value is 0 (zero).

Any interface-level DSCP rewrite rules you have configured override the DSCP value you specify for the export profile. You need to specify a DSCP value for the export

profile only if you do not configure DSCP rewrite rules on the outgoing interface. For more information, see *Configuring Rewrite Rules*.

```
[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 export-params]
user@host# set forwarding-class assured forwarding
```

9. (Optional) (MX Series routers only on Junos OS Release 17.3R1 or later) Specify a packet loss priority to assign to exported packets.

```
[edit services analytics export-profile name]
user@host# set loss-priority (low | high | medium-low | medium-high)
```

For example, to specify a loss priority of high for an export profile with the name export-params:

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

Configure the Data Collector

1. Ensure the collector is ready to receive data from the Juniper device:
 - IP Address: The device's management IP (192.168.1.100).
 - Port: The configured port (for example, 2000 for UDP, 50051 for gRPC).
 - Protocol: Match the transport protocol (for example, UDP or gRPC).
 - Credentials: If TLS is enabled, configure the collector with the appropriate client certificate and key.
 - Data format: Ensure the collector supports Google Protocol Buffers (gpb) and has the necessary protobuf definitions to decode sensor data.

Example collector configuration (for a UDP-based collector):

Configure the collector to listen on 192.168.1.100:2000 and process gpb-encoded data.

2. On the data collector, verify if the data is received. For example, if Prometheus or Grafana is used as a data collector, check the logs or visualizations. For custom collectors, verify the data stream matches the configured sensor paths.
3. To decode Junos Telemetry Interface Data, see [Decoding Junos Telemetry Interface Data With UNIX Utilities](#).

Verify the Junos Telemetry Interface Sensor Configuration

IN THIS SECTION

- [Purpose | 103](#)
- [Action | 104](#)

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;
    loss-priority high;
    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       : 192.0.2.2
Remote-port          : 30000

```

Profile Information :

```

Name                : export-params
Rep-interval         : 20
Address              : 192.0.2.3
Port                 : 21111
Timestamp            : 1
Format               : GPB
Transport             : UDP
DSCP                  : 20
Forwarding-class      : assured-forwarding
Loss-priority         : high

```



NOTE: The `show agent sensors` command output for gRPC sensors is truncated on the Junos OS Evolved platform to align with the output format of the Junos OS platform.

Selecting Sensor Paths

IN THIS SECTION

- [Sensor Explorer and Guidelines for Selecting Sensor Paths | 105](#)

Sensor Explorer and Guidelines for Selecting Sensor Paths

Use the Juniper Networks [Junos YANG Data Model Explorer](#) to view all the supported resource paths, their corresponding leaves, and the device platforms that support them.



NOTE: The

[Junos YANG Data Model Explorer](#)

was introduced in the 23.2R2-S2 releases. From releases 20.2R1 up to 23.1R1, the sensor information is available in the

[Junos Telemetry Sensor Explorer](#)

.

To search and view other telemetry sensors and specific information about some legacy sensors, see ["Legacy Sensor Paths" on page 255](#).

For information on guidelines and best practices for configuring sensor paths, see ["Explore Sensor Paths" on page 9](#).

Decoding Junos Telemetry Interface Data With UNIX Utilities

SUMMARY

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.

IN THIS SECTION

- [Preparing the Collector to Decode Data | 106](#)
- [Decoding Data on the Collector | 107](#)
- [Decoding Junos Telemetry Interface UDP Data at the Collector | 119](#)

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: <https://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 < ../data.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 Google protocol buffers (gpb) library.

```

protoc --decode TelemetryStream logical_port.proto -I /usr/include -I . < data.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"
      }
    }
  }
}
}
}

```

Decoding Junos Telemetry Interface UDP Data at the Collector

The collector must start a listener on the remote address or port combination to read the incoming data. The incoming data can be decoded using the *telemetry_top.proto* and the sensor proto files.

To simplify the decoding procedure, the collectors can load all the proto files shipped as part of the Telemetry Software package to decode the incoming data.



NOTE:

1. Verify streaming data on both management and WAN interfaces. Specify the appropriate address in the streaming server profile.
2. Enums and float are streamed as strings for UDP streaming. Enums will be retained as strings as gNMI uses the same format.
3. The handling of float data type is scoped for a future release.

When telemetry data is streamed over UDP, ensure that data is decoded correctly, and unknown fields are not decoded at the collector.

Below are some issues observed while streaming telemetry data over UDP:

1. **Incorrect Decoding:** Incorrect decoding implies that telemetry data is not correctly streamed over UDP. In the following example, the data highlighted in bold indicates that the data was not decoded correctly. This behaviour is either due to incorrect encoding or wrong proto file packaging.

```

system_id: "r02.dtw01.icn"
component_id: 65535
sensor_name: "mpls:/network-instances/network-instance/mpls:/network-instances/network-
instance/mpls:rpd"
sequence_number: 2421 timestamp: 1715024560793
version_major: 1 version_minor: 0 enterprise: { [juniperNetworks]:
{ [jnpr_network_instances_rsvp_ext]: {
  network_instance: { name: "master" mpls: { 153 { 151
{      152{      151{      51: "r02.dtw01.icn-r01.bos02.icn-01"
      152{      151{      51: "AUTO"
      52: 0
      53: 12873154
    }      152{
      151{
      51:1
      52:200000
      53: 0
      54: 300
      55: 1

      61: 1549061
    }      152 {

```

Below is the example of the data being decoded correctly:

```

system_id: "r0-RE0"
component_id: 65535
sub_component_id: 0
sensor_name: "test_chassisd:/network-instances:/network-instances:/rpd"
sequence_number: 0
timestamp: 1719126223900
version_major: 1
version_minor: 0
enterprise {
  [juniperNetworks] {

```



```
[jnpr_network_instances_ni_226_ext] {
  network_instance {
    name: "DEFAULT"
    protocols {
      protocol {
        identifier: "STATIC"
        name: "DEFAULT"
        static_routes {
          static {
            prefix: "10.0.0.0/8"
            state {
              prefix: "10.0.0.0/8"
              set_tag: "0"
            }
          }
          next_hops {
            next_hop {
              index: "1"
              state {
                index: "1"
                next_hop: "10.220.127.254"
                metric: 0
                recurse: false
              }
            }
            interface_ref {
              state {
                interface: "re0:mgmt-0"
                subinterface: 0
              }
            }
          }
        }
      }
    }
  }
}
```

- 2. Incomplete data streaming:** Streaming of incomplete data must be verified through data validation in the output file.

RELATED DOCUMENTATION

| [Configuring a Junos Telemetry Interface Sensor \(CLI Procedure\)](#)

4

PART

Junos Telemetry Interface Features

- [gNMI Genstate Subscription | 124](#)
- [gRPC Tunnels Overview | 126](#)
- [End-of-Message Notification for Routing Engine Sensor | 133](#)
- [Enabling Client Streaming and Bidirectional Streaming of Telemetry Sensor Information | 134](#)
- [Configure a NETCONF Proxy Telemetry Sensor in Junos | 135](#)
- [Support for Zero Suppression | 151](#)
- [Configure SR-TE for Uncolored Ingress LSPs | 151](#)
- [Per-Path or Per-Segment-List Traffic Statistics | 154](#)
- [SRv6 and SRv6-TE Traffic Sensor Telemetry | 159](#)
- [On-Box Aggregation Overview | 163](#)
- [Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets | 169](#)
- [Enabling Export of Transit SPRING Statistics | 179](#)
- [Configuring IP-AFT Prefix Filtering | 184](#)
- [Enable Subscriber and Service Accounting for BNG CUPS Telemetry | 185](#)
- [CPU and NPU Sensor Support for MX Series Routers with MPC10E-15C-MRATE Line Cards | 189](#)
- [Delivery of Telemetry Data for AFT-Based Line Cards on MX Series Routers | 189](#)
- [Diameter Application Protocol And Diameter Peer Sensors For Subscribers | 190](#)
- [Dynamic Tunnel Statistics Support | 191](#)
- [Enabling Streaming of Telemetry Sensor Information for SR-TE policies \(BGP or Static\) | 192](#)

- [Exporting Packet Forwarding Engine Traffic Sensor Data | 193](#)
 - [Export Timing Data to Collectors | 195](#)
 - [FPC and Optics Support | 197](#)
 - [Interface Burst Monitoring | 198](#)
 - [Interface Express Sensor | 198](#)
 - [JTI Broadband Edge Statistics Support for Junos Fusion on MX Series | 199](#)
 - [Physical Ethernet Interface Sensor | 200](#)
 - [Sensor Power-State Management Support Using gNMI | 200](#)
 - [Standby Routing Engine Sensors For Subscribers | 203](#)
 - [Support For LSP Statistics | 204](#)
 - [NPU Sensors | 207](#)
 - [Telemetry Streaming Support for ZR/ZR+ Optics on MPC 10 Linecards | 223](#)
 - [Leaf-Level Support for PFE Sensors | 224](#)
 - [Resource Filtering | 227](#)
 - [Transceiver Diagnostics | 233](#)
 - [VLAN Sensors | 233](#)
-

gNMI Genstate Subscription

A gNMI Genstate subscription utilizes the gNMI protocol to retrieve state data from a network device, specifically focusing on data exposed through Genstate YANG data models. This allows you to monitor the operational state of the device, similar to data presented in CLI "show" commands, but through a streaming mechanism.

The Genstate model allows you to access operational state information in a structured, machine-readable format. The network device, acting as a gNMI server, sends telemetry data corresponding to the subscribed paths. This data can be used for monitoring, alerting, and other applications.

gNMI clients in gRPC dial-in environments can subscribe to a Genstate resource path and STREAM statistics in SAMPLE sub mode to a collector for telemetry data.



NOTE: ON_CHANGE sub mode is not supported. For more information, see ["Guidelines for Telemetry Data Subscriptions over gNMI" on page 29](#).

Use the following steps to set up gNMI Genstate subscription:

1. View or download the models from the [Juniper/yang](#) GitHub repository or [Juniper Networks](#).
2. Obtain and load the gRPC server's certificate.
3. Enable your network device for gRPC services.
4. Specify the local certificate used for server authentication.



NOTE: For Genstate subscription step details, see [Configure gRPC Services](#).

The subscription example that follows uses a target device configured for mutual authentication using X.509 certificates. The subscription request supplies the root CA certificate required to verify the target device's local certificate, and it also provides the client's key and certificate so that the target device can authenticate the client. The subscription uses STREAM mode and submode SAMPLE and requests data for the resource path `genstate:/genstate/system-information/os-version`. The telemetry data returned to the collector shows that the target device is operating with Junos OS version 24.2R1.12-EV0.



NOTE: Junos OS supports several origin schemas including Juniper Native, OpenConfig, and Genstate. The origin is stated in the path before the colon where **genstate** is the

origin in the path `genstate:/genstate/system-information/os-version`. For more information on origin in path, see ["gNMI Service" on page 26](#).

```
user@controller:~$ gnmic sub -a 198.51.100.10:32767 --mode stream --stream-mode sample -u grpc-
user -p secret --tls-ca certs/serverRootCA.crt --tls-cert certs/client.crt --tls-key certs/
client.key --sample-interval 30s --path genstate:/genstate/system-information/os-version
{
  "source": "198.51.100.10:32767",
  "subscription-name": "default-1708559603",
  "timestamp": 1708559609668499930,
  "time": "2024-02-21T15:53:29.66849993-08:00",
  "prefix": "genstate:genstate/system-information/os-version",
  "updates": [
    {
      "Path": "os-version",
      "values": {
        "os-version": "24.2R1.12-EV0"
      }
    }
  ]
}
```

The subscription example that follows again uses a target device configured for mutual authentication using X.509 certificates. The subscription uses STREAM mode and submode SAMPLE and requests data for the resource path `genstate:/genstate/interface-information/physical-interface[name=et-1/0/1]/admin-status`. The admin-status (administrative status) indicates whether the interface is enabled (up) or disabled (down). The telemetry data returned to the collector shows that physical interface `et-1/0/1` is up.

```
user@controller:~$ gnmic sub -a 198.51.100.10:32767 --mode stream --stream-mode sample -u grpc-
user -p secret --tls-ca certs/serverRootCA.crt --tls-cert certs/client.crt --tls-key certs/
client.key --sample-interval 30s --path genstate:/genstate/interface-information/physical-
interface[name=et-1/0/1]/admin-status
{
  "source": "198.51.100.10:32767",
  "subscription-name": "default-1708561308",
  "timestamp": 1708561313983469233,
  "time": "2024-02-21T16:21:53.983469233-08:00",
  "prefix": "genstate:genstate/interface-information/physical-interface[name=et-1/0/1]/admin-
status",
  "updates": [
```

```
{
  "Path": "admin-status",
  "values": {
    "admin-status": "up"
  }
}
]
```

Change History Table

Feature support is determined by the platform and release you are using. Use [Feature Explorer](#) to determine if a feature is supported on your platform.

Release	Description
15.1F3	Junos OS Evolved Release 24.2R1, Juniper Networks publishes the <i>genstate</i> YANG data models. The genstate models are subscribable YANG models for operational state data on Junos devices. gNMI clients can subscribe to resource paths defined in the models to retrieve specific state data.

RELATED DOCUMENTATION

| [Junos Genstate YANG Data Models](#)

gRPC Tunnels Overview

IN THIS SECTION

- [Example: Configure a gRPC Tunnel](#) | 128

A gRPC tunnel acts as a client-server protocol that dials out a session from the target to the TCP client through the default routing instance or configured routing instance. If you do not choose to configure a routing instance, the gRPC tunnel uses the default routing instance.

You can configure the source address for each gRPC tunnel session to dial out a connection to the tunnel server. If you do not configure the source address, the kernel picks the source address that can reach the tunnel server.

For more information about gRPC tunnels, see <https://github.com/openconfig/grpctunnel>.

A gRPC tunnel has three main entities:

- **Target**— Represents the network device. The target is a gRPC client.
- **Tunnel Server**— A software entity that is an off-box application that manages the subscription and target registrations. The tunnel server is a gRPC server.
- **Tunnel Client**— A software entity that performs client tasks. The tunnel client may be self-contained within the tunnel server. The tunnel client is also a gRPC client.

Benefits of using a gRPC tunnel session:

- Overcomes a series of complex firewall configurations as the connection is established from the server-side.
- Access various TCP server applications such as gNMI-gNOI, SSH, or NETCONF-SSH without any operational requirements.

gRPC Tunnel Architecture

A gRPC tunnel is a generic infrastructure for TCP-based applications to communicate within gRPC messages.

In general, the TCP client initiates a connection to the TCP server or target. Junos devices act as target that runs TCP server applications such as gNMI-gNOI, SSH, and NETCONF-SSH. If a client is unable to reach the server, common reasons could be:

- The server runs into complex firewall configurations with firewalls preventing inbound connections.
- The server runs into a router implementing network address translation (NAT).
- The server comes across any other operational requirements, which prevents outside connections.

When a TCP client is not able to reach a target, you can configure a gRPC tunnel session to establish a connection between the TCP client and the target. A gRPC tunnel session establishes a connection in the reverse direction where a target dials out to a TCP client.

To use a gRPC tunnel session, a tunnel client is added to the target-side where the `grpc-tunnel` process runs and all the gRPC tunnel-related configurations are performed. On the TCP client-side, a tunnel server is added.

You must include the `grpc-tunnel` configuration statement in the `[edit system services]` hierarchy to configure a gRPC tunnel session.

gRPC Tunnel Security

The gRPC tunnel is a dial-out model where a device initiates a connection based on the configuration. The gRPC tunnel is on a secure gRPC channel that uses TLS certificates.

Example: Configure a gRPC Tunnel

IN THIS SECTION

- [Overview | 128](#)
- [Requirements | 128](#)
- [Topology | 128](#)
- [Configure a gRPC tunnel | 129](#)

Overview

This section covers the steps needed to configure the target for this example. The focus is on the target because that is the Junos device where the gRPC tunnel is configured.

Requirements

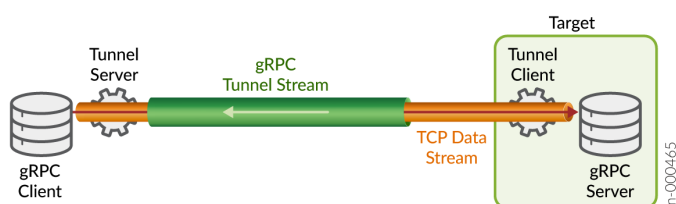
This example uses the following software and hardware components:

- Junos OS or Junos Evolved Release 22.4 or later for routing and switching devices
- One host device as the tunnel client
- One host device as the tunnel server

Topology

[Figure 1 on page 129](#) shows the topology used in this example.

Figure 5: gRPC tunnel setup



Based on the Junos configuration of the target device, it dials-out a `grpc-tunnel` to the tunnel server. The target device registers itself with the tunnel server through the Register stream RPC.

When a client requests a TCP session to a specific target, the tunnel server acts as an intermediary and connects to the above registered target session on tunnel server.

If the target supports the requested target type, the device will dial-out a new tunnel called the Tunnel stream RPC. This establishes a gRPC tunnel between the client and the target through the tunnel server. The tunnel client can now access the intended TCP applications on the target.



NOTE: There can only be one Register stream between a tunnel server and the network device but multiple Tunnel streams for the same pair.

Configure a gRPC tunnel

IN THIS SECTION

- [CLI Quick Configuration | 129](#)
- [Step-by-Step Procedure | 130](#)
- [Results | 132](#)

CLI Quick Configuration

To quickly configure this example, copy the following commands, paste them into a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the [edit] hierarchy level.

```
set system services grpc-tunnel servers server server1 address 10.205.0.1
set system services grpc-tunnel servers server server1 port 50301
```

```
set system services grpc-tunnel servers server server1 credentials tls certificate-id client-
cert1
set system services grpc-tunnel servers server server1 targets ssh
```

Step-by-Step Procedure

Follow these steps to configure the gRPC tunnel in the target device.

1. Configure the servers under gRPC-tunnel.

- a. Configure the IPv4 or IPv6 address or hostname of the tunnel server.**

```
[edit system services]
user@host# set grpc-tunnel servers server server1 address 10.205.0.1
```

- b. Configure the port number through which the tunnel server listens.**

```
[edit system services]
user@host# set grpc-tunnel servers server server1 port 50301
```

- c. Configure the credentials using the **tls** statement.**

```
[edit system services]
user@host# set grpc-tunnel servers server server1 credentials tls certificate-id client-
cert1
```

- d. Specify the target applications you want to access. The options available are **ssh**, **netconf-ssh** and **gnmi-gnoi**.**

```
[edit system services]
user@host# set grpc-tunnel servers server server1 targets ssh
```

2. (Optionally) Set the **retry-interval (in seconds). If the tunnel-server is unreachable, the target device retries to connect after the retry-interval.**

```
[edit system services]
user@host# set grpc-tunnel servers retry-interval 30
```

3. (Optionally)

Set the **routing instance**. If you do not set the routing instance, the gRPC tunnel uses the default routing instance.

```
[edit system services]
user@host# set grpc-tunnel servers server ts1 routing-instance routing-instance
```

4. (Optionally)

Set the **source address**. If you do not set the source address, the kernel picks the source address that can reach the tunnel server.

```
[edit system services]
user@host# set grpc-tunnel servers server ts1 source-address 69.70.31.677
```

5. (Optionally) Configure the **target-string-option** under **grpc-tunnel**.

- a. Use the **pattern** statement to create an ordered list of supported options.

```
[edit system services]
user@host# set grpc-tunnel target-string-option pattern hostname custom
```

- b. Use the **custom-string** statement to define a custom string that is sent when the statement **pattern** contains **custom** as one of the options.

```
[edit system services]
user@host# set grpc-tunnel target-string-option custom-string device1
```

- c. Use the **delimiter** statement when more than one option is selected in the pattern. By default, the | (pipe symbol) is used.

```
[edit system services]
user@host# set grpc-tunnel target-string-option delimiter |
```



NOTE: A maximum of 10 tunnel servers can be configured.

Results

Display the results of the configuration on the target device. The output reflects only the functional configuration added in this example.

```
user@host> show configuration system services
```

```
grpc-tunnel {
  servers {
    retry-interval 30;
    server server1 {
      address 10.205.0.01;
      port 50301;
      credentials {
        tls {
          certificate-id client-cert1;
        }
      }
      targets [ ssh ];
      routing-instance routing-instance;
      source-address 69.70.31.677;
    }
    server server2 {
      address 10.205.0.02;
      port 50302;
      credentials {
        tls {
          ca-profiles [ serverRootCA1 serverRootCA2 ];
        }
      }
      targets [ gnmi-gnoi netconf-ssh ];
      routing-instance routing-instance;
      source-address 69.70.31.677;
    }
  }
}
target-string-option {
  pattern [ hostname vendor model version custom ];
  custom-string device1;
  delimiter |;
```

```
}
}
```

RELATED DOCUMENTATION

| *grpc-tunnel*

End-of-Message Notification for Routing Engine Sensor

Starting in Junos OS Release 21.2R1, we've introduced an end-of-message (EoM) Boolean flag for all JT1 Routing Engine sensors. The flag notifies the collector that the current wrap has completed for a particular sensor path. A wrap is a complete key-value data dump for all the leaves under a sensor path.

The EoM flag also enables the collector to detect when the end of wrap occurs without having to compare stream creation timestamp values that the collector receives from the packets. Comparing timestamp values is costly time-wise and delays data aggregation.

To use this feature with gRPC Network Management Interface (gNMI) transport or Remote Procedure Call (gRPC), retrieve the protobuf files from the relevant branch on the [Juniper Networks](#) download site:

- GnmiJuniperTelemetryHeaderExtension.proto (gNMI)
- agent.proto (for gRPC)

For example: <https://github.com/Juniper/telemetry/blob/master/20.3/20.3R1/protos/GnmiJuniperTelemetryHeaderExtension.proto>.

After you download and install the new protobuf files on a collector, the EoM field is present in the packets received.

Enabling Client Streaming and Bidirectional Streaming of Telemetry Sensor Information

Starting with Junos OS Release 18.1R1, OpenConfig support through Remote Procedure Calls (gRPC) and JTI is extended to support client streaming and bidirectional streaming of telemetry sensor information on MX Series and PTX Series routers.

APIs are implemented in Junos based on Protobuf specifications for OpenConfig. These APIs perform configuration, operational state retrieval, and telemetry on Junos routers using gRPC as the transport mechanism.

With client streaming, the client sends a stream of requests to the server instead of a single request. The server typically sends back a single response containing status details and optional trailing metadata. With bidirectional streaming, both client and server send a stream of requests and responses. The client starts the operation by invoking the RPC and the server receives the client metadata, method name, and deadline. The server can choose to send back its initial metadata or wait for the client to start sending requests. The client and server can read and write in any order. The streams operate completely independently.

Junos devices can be managed through API (RPC) prototypes:

- rpc Capabilities (CapabilityRequest)

Returns (CapabilityResponse). Allows the client to retrieve the set of capabilities that is supported by the target.

- rpc Get (GetRequest)

Returns (GetResponse). Retrieves a snapshot of data from the target.

- rpc Set (SetRequest)

Returns (SetResponse). Allows the client to modify the state of data on the target.

- rpc Subscribe (stream SubscribeRequest)

Returns (stream SubscribeResponse). Allows a client to request the target to send it values for particular paths within the data tree. These values may be streamed (STREAM) or sent one-off on a long-lived channel (POLL), or sent as a one-off retrieval (ONCE). If a subscription is made for a top-level container with a sample frequency of 0, leaves with ON_CHANGE support are streamed based on events. Other leaves will not be streamed.

Juniper Extension Toolkit (JET) support provides insight to users regarding the status of clients connected to JSD. JET support for gRPC includes expanding the maximum number of clients that can connect to JSD from 8 to 30 (the default remains 5). To specify the maximum number of connections,

include the `max-connections` statement at the `[edit system services extension-service request-response grpc]` hierarchy level.

To provide information regarding the status of clients connected to JSD, issue the enhanced `show extension-service client information` command and include the `clients` or `servers` options. The `clients` option displays request-response client information. The `servers` option displays request-response server information.

Configure a NETCONF Proxy Telemetry Sensor in Junos

IN THIS SECTION

- [Create a User-Defined YANG File | 139](#)
- [Load the Yang File in Junos | 144](#)
- [Collect Sensor Data | 145](#)
- [Installing a User-Defined YANG File | 148](#)
- [Troubleshoot Telemetry Sensors | 149](#)

Using Junos telemetry streaming, you can turn any available state information into a telemetry sensor by means of the XML Proxy functionality. The NETCONF XML management protocol and Junos XML API fully document all options for every supported Junos OS operational request. After you configure XML proxy sensors, you can access data over NETCONF “get” remote procedure calls (RPCs).

This task shows you how to stream the output of a Junos OS operational mode command.



BEST PRACTICE: We recommend not to use YANG files that map to a Junos OS operational command with extensive or verbose output or one that is slow in producing output. Commands with a noticeable delay should be avoided in YANG files. Including such commands can affect other `xmlproxyd` sensors as well as the performance of `xmlproxyd`.

The output from some operational mode commands is dynamic and the level of their verbosity depends on factors such as the configuration and hardware. Examples of such commands include any variation of `show interfaces`, `show route`, `show arp`, `show bfd`, `show bgp`, and `show ddos-protection`.

To check the verbosity level of a command, issue the ***command-name* | display xml | count** command. If the line count exceeds a value of 4000 lines, then the command is not recommended for XML proxy streaming. This value is more of an approximation based on internal base-lining. It can be less depending upon various factors such as device type, processing power of the device, and the existing CPU load. Consequently, this feature needs to be used judiciously based on how the device is performing.

You can issue the command ***command-name* | display xml** before using a YANG file that maps to a Junos OS or Junos OS Evolved operational mode command to verify that the command produces valid XML output and does not contain invalid tags, data, or formatting.

Using a YANG file that maps to a verbose command results in one or more of following:

- The `xmlproxyd` process CPU utilization remains high. If `xmlproxyd` has tracing enabled, the CPU utilization is even higher.
- An increase in the `xmlproxyd` process memory utilization.
- The `xmlproxyd` process state may show `sbwait`, indicating that the command output is verbose and that `xmlproxyd` is spending significant time reading the command's remote procedure call's (RPC's) output.
- The `xmlproxyd` sensor data does not complete the wrap.
- The `xmlproxyd` streams partial or no data for the sensors.
- The `xmlproxyd` misses reporting-interval cycles. The intervals start to overlap because of a command's verbose output, resulting in the `xmlproxyd`'s sensor streaming data that is slow or delayed.
- The process or application that serves the verbose command's RPC may show high CPU numbers or delays in performing main tasks. This behavior is caused when the process or application is busy serving the RPC that has verbose output.

This task requires the following:

- An MX Series, vMX Series, or PTX Series router operating Junos OS Release 17.3R2 or later.
- A telemetry data receiver, such as OpenNTI, to verify proper operation of your telemetry sensor.

In this task, you will stream the contents of the Junos OS command `show system users`.

show system users (vMX Series)

```
user@switch> show system users
```

USER	TTY	FROM	LOGIN@	IDLE	WHAT
user1	pts/0	172.31.12.36	12:40PM	39	-cli (cli)
user2	pts/1	172,16.03.25	3:01AM	-	-cli (cli)

In addition to the expected list of currently logged-in users, the `show system users` output also provides the average system load as 1, 5 and 15 minutes. You can find the load averages by using the `show system users | display xml` command to view the XML tagging for the output fields. See `<load-average-1>`, `<load-average-5>`, and `<load-average-15>` in the XML tagging output below.

```
user@switch> show system users | display xml

<rpc-reply xmlns:junos="http://xml.juniper.net/junos/17.4R1/junos">
  <system-users-information xmlns="http://xml.juniper.net/junos/17.4R1/junos">
    <uptime-information>
      <date-time junos:seconds="1520170982">1:43PM</date-time>
      <up-time junos:seconds="86460">1 day, 40 mins</up-time>
      <active-user-count junos:format="2 users">2</active-user-count>
      <load-average-1>0.70</load-average-1>
      <load-average-5>0.58</load-average-5>
      <load-average-15>0.55</load-average-15>
    <user-table>
      <user-entry>
        <user>root</user>
        <tty>pts/0</tty>
        <from>172.21.0.1</from>
        <login-time junos:seconds="1520167202">12:40PM</login-time>
        <idle-time junos:seconds="0">-</idle-time>
        <command>cli</command>
      </user-entry>
      <user-entry>
        <user>mwiget</user>
        <tty>pts/1</tty>
        <from>66.129.241.10</from>
        <login-time junos:seconds="1520170862">1:41PM</login-time>
        <idle-time junos:seconds="60">1</idle-time>
        <command>cli</command>
      </user-entry>
    </user-table>
  </system-users-information>
</rpc-reply>
```

```

        </user-table>
    </uptime-information>
</system-users-information>
<cli>
    <banner></banner>
</cli>
</rpc-reply>

```



TIP: The uptime-information tag shown in the preceding output is a container that contains leafs, such as date-time, up-time, active-user-count. and load-average-1. Below is a sample YANG file for this container:

```

container uptime-information {
    dr:source "uptime-information"; // Exact name of the XML tag
    leaf date-time { // YANG model leaf
        type string; // Type of value
        dr:source date-time; // Exact name of the XML tag
    }
    leaf up-time { // YANG model leaf
        type string; // Type of value
        dr:source up-time; // Exact name of the XML tag
    }
    leaf active-user-count { // YANG model leaf
        type int32; // Type of value
        dr:source active-user-count; // Exact name of the XML tag
    }
    leaf load-average-1 { // YANG model leaf
        type string; // Type of value
        dr:source load-average-1; // Exact name of the XML tag
    }
    ...
}

```



TIP: The uptime-information tag also has another container named user-table that contains a list of user entries.
Below is a sample YANG file for this container:

```

container user-table { // "user-table" container which contains list of user-entry
    dr:source "user-table"; // Exact name of the XML tag
}

```

```

list user-entry { // "user-entry" list which contains the users' details in form of leafs
  key "user"; // Key for the list "user-entry" which is a leaf in the list "user-entry"
  dr:source "user-entry"; // Source of the list "user-entry" which is the exact name of
the XML tag
  leaf user { // YANG model leaf
    dr:source user; // A leaf in the list "user-entry", exact name of the XML tag
    type string; // Type of value
  }
  leaf tty { // YANG model leaf
    dr:source tty; // A leaf in the list "user-entry", exact name of the XML tag
    type string; // Type of value
  }
  leaf from { // YANG model leaf
    dr:source from; // A leaf in the list "user-entry", exact name of the XML tag
    type string; // Type of value
  }
  leaf login-time { // YANG model leaf
    dr:source login-time; // A leaf in the list "user-entry", exact name of the XML tag
    type string; // Type of value
  }
  leaf idle-time { // YANG model leaf
    dr:source idle-time; // A leaf in the list "user-entry", exact name of the XML tag
    type string; // Type of value
  }
  leaf command { // YANG model leaf
    dr:source command; // A leaf in the list "user-entry", exact name of the XML tag
    type string; // Type of value
  }
}
}

```

Create a User-Defined YANG File

The YANG file defines the Junos CLI command to be executed, the resource path the sensors are placed under, and the key value pairs taken from the matching XML tags.

Custom YANG files for Junos OS conform to the YANG language syntax defined in RFC 6020 YANG 1.0 *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)* and RFC 7950

The YANG 1.1 Data Modeling Language. Certain directives need to be present in the file that configure XML proxy.

To use the `xmlproxyd` process to translate telemetry data, create a `render.yang` file. In this file, the `dr:command-app` is set to `xmlproxyd`.

The XML proxy YANG filename and module name must start with `xmlproxyd_`:

- For the XML proxy YANG filename, add the extension `.yang`, for example, `xmlproxyd_sysusers.yang`
- For the module name, use the filename without the extension `.yang`, for example, `xmlproxyd_sysusers`

To simplify creating a YANG file, it's easiest to start by modifying a working example.

1. Provide a name for the module. The module name must start with `xmlproxyd_` and be the same name as the XML proxy YANG file name.

For example, for an XML proxy YANG file called `sysusers.yang`, drop the `.yang` extension and name the module `xmlproxyd_sysusers`:

```
module xmlproxyd_sysusers {
```

2. For the Junos telemetry interface include the process name `xmlproxyd`:

```
dr:command-app "xmlproxyd";
```

3. Include the following RPC for the NETCONF get request:

```
rpc juniper-netconf-get {
```

4. Specify the location of the output of the RPC, where *company-name* is the name you give to the location:

```
dr:command-top-of-output "/company-name";
```

5. Include the following command to execute the RPC:

```
dr:command-full-name "drend juniper-netconf-get";
```

6. Specify the CLI command from which to retrieve data. The Junos OS CLI command that gets executed at the requested sample frequency is defined under `dr:cli-command` and executed by the `xmlproxyd` process.

To retrieve command output for the Junos OS command `show system users`:

```
dr:cli-command "show system users";
```

7. Escalate privileges, logon as "root", connect to the internal management socket via Telnet, and specify help for an RPC:

```
dr: command-help "default <get> rpc";
```

When this is included in the YANG file, output that is helpful for debugging is displayed in the help drend output on the internal management socket:

```
telnet /var/run/xmlproxyd_mgmt
Trying /var/run/xmlproxyd_mgmt...
Connected to /var/run/xmlproxyd_mgmt.
Escape character is '^]'.
220 XMLPROXYD release 18.2I20180412_0904_bijchand built by bijchand on 2018-04-12 14:48:48 UTC
help drend
```

200-juniper-netconf-get-0 system users <get> RPC

8. Specify the hierarchy and use the `dr:source` command to map to a container, a list, or a specific leaf. The absolute path under which the sensors will be reported is built from the output group `junos` plus `system-users-information`, concatenated by `/`. The path `/junos/system-users-information/` is the path to query for information about this custom sensor.



WARNING: You should not create a custom YANG model that conflicts or overlaps with predefined native paths (Juniper defined paths) and OpenConfig paths (resources). Doing so can result in undefined behavior.

For example, do not create a model that defines new leafs at or augments nodes for resource paths such as `/junos/system/linecard/firewallor /interfaces`.

A one-to-one mapping between container, leafs and the XML tag or value from the CLI command output is defined in the grouping referenced by `uses` within the output container. A *grouping* can be referred to multiple times in different container outputs. The container `system-users-information` below uses the grouping `system-users-information`. However, it is defined without the aforementioned one-to-one mapping for every container, list and leaf to an output XML tag from the CLI command XML output.

```
output {
  container junos {
    container system-users-information {
      dr:source "/system-users-information";
      uses system-users-information-grouping;
    }
  }
}
```

9. The following YANG file shows how to include these commands to enable the `xmlproxyd` process to retrieve the full operational state and map it to the leafs in Juniper's own data model:

```

*/

/*
 * Example yang for generating OpenConfig equivalent of show system users
 */

module xmlproxyd_sysusers {
  yang-version 1;

  namespace "http://juniper.net/yang/software";

  import dend {
    prefix dr;
  }

  grouping system-users-information-grouping {
    container uptime-information {
      dr:source "uptime-information";
      leaf date-time {
        type string;
        dr:source date-time;
      }
      leaf up-time {
        type string;
        dr:source up-time;
      }
      leaf active-user-count {
        type int32;
        dr:source active-user-count;
      }
      leaf load-average-1 {
        type string;
        dr:source load-average-1;
      }
      leaf load-average-5 {
        type string;
        dr:source load-average-5;
      }
    }
  }
}

```

```

leaf load-average-15 {
    type string;
    dr:source load-average-15;
}
container user-table {
    dr:source "user-table";
    list user-entry {
        key "user";
        dr:source "user-entry";
        leaf user {
            dr:source user;
            type string;
        }
        leaf tty {
            dr:source tty;
            type string;
        }
        leaf from {
            dr:source from;
            type string;
        }
        leaf login-time {
            dr:source login-time;
            type string;
        }
        leaf idle-time {
            dr:source idle-time;
            type string;
        }
        leaf command {
            dr:source command;
            type string;
        }
    }
}
}

dr:command-app "xmlproxyd";
rpc juniper-netconf-get {
    dr:command-top-of-output "/company-name";
    dr:command-full-name "drend juniper-netconf-get";
    dr:cli-command "show system users";
}

```

```

dr:command-help "default <get> rpc";
output {
  container company-name {
    container system-users-information {
      dr:source "/system-users-information";
      uses system-users-information-grouping;
    }
  }
}
}
}
}

```

Load the Yang File in Junos

After the YANG file is complete, upload the YANG file and verify that the module is created.

1. Upload the YANG file to the router.
2. Register the YANG file using the request `system yang add package` command.

```

user@switch> request system yang add package sysusers proxy-xml module xmlproxyd_sysusers.yang
XML proxy YANG module validation for xmlproxyd_sysusers.yang : START
XML proxy YANG module validation for xmlproxyd_sysusers.yang : SUCCESS
JSON generation for xmlproxyd_sysusers.yang : START
JSON generation for xmlproxyd_sysusers.yang: SUCCESS

```



NOTE: Starting in Junos OS Release 18.3R1, adding, deleting, or updating YANG packages in configuration mode with the `run` command is not supported.

3. Verify that the module (sensor) is registered using the `show system yang package sysusers` command, where `sysusers` is the name of the package:

```

user@switch> show system yang package sysusers
Package ID           :sysusers
XML Proxy YANG Module(s) :xmlproxyd_sysusers.yang

```


4. Enable gRPC in the Junos OS configuration:

```
user@switch> set system services extension-service request-response grpc port 32767
```

Collect Sensor Data

IN THIS SECTION

- [Platform-Specific Collecting Sensor Data Behavior | 145](#)

Use your favorite collector to pull the newly created telemetry sensor data from the device.

Consider resource constraints before initiating sensors:

- Avoid specifying the same reporting interval for multiple XML proxy sensors.
- Because xmlproxyd performs XML and text processing, a device should only contain XML proxy sensors that execute within the CPU utilization range.

Platform-Specific Collecting Sensor Data Behavior

Use [Feature Explorer](#) to confirm platform and release support for specific features.

Use the following table to review platform-specific behaviors for your platforms:

Table 8: Collecting Sensor Data Behavior

Platform	Difference
PTX10008	For PTX10008 routers operating Junos OS Evolved, do not connect more than 10 collectors per router for telemetry RPCs.

The following instructions use the collector *jtimon*. For information about jtimon setup, see [Junos Telemetry Interface client](#).



NOTE: If a subscription already exists for a sensor and a duplicate subscription is configured, the connection between the collector and the device will close with the error message `AlreadyExists`.

1. Create a simple configuration file, here named `vmx1.json`. Adjust the host IP address and the port, as needed. The path `/junos/system-users-information` is specified. The `freq` field is defined in MicroSoft, streaming a new set of key value pairs every 5 seconds. Optionally, you can add multiple paths.

```
$ cat vmx1.json
{
  "host": "172.16.122.182"
  "port": 32767
  "cid": "my-client-id",
  "grpc" : {
    "ws" : 524289
  },
  "paths": {
    {
      "path": "/junos/system-users-information/",
      "freq": 5000
    },
    {
      "path": "/junos/additional-path/", <-OPTIONAL
      "freq": 5000
    }
  }
}
```

2. Launch the collector, using either your own compiled file or an automatically built image from Docker Hub. The sample query output below shows the sensor report by path. Every key is sent in human-readable form as an absolute path. In case of lists, the absolute path contains an index in the form of `XPATH` which is ideal to group values from a (time series) database, such as InfluxDB. For example, the output below shows the path `/junos/system-users-information/uptime-information/user-table/user-entry[user='ab']/`.

You can terminate the stream of sensor data using `Ctrl-C`.

```
$ docker run -tu --rm -v $(PWD):/u mw/jtimon --config vmx1.json --print
gRPC headers from Junos:
  init-response: [response { subscription_id 1} path_list {path: "junos/system-users-
information/" sample-frequency: 5000 } ]
```

```

content-type: [application/grpc]
grpc-accept-encoding: [identity,deflate,gzip]
2018/03/04 17:13:19 system-id vmxdockerlight_vmx1_1
2018/03/04 17:13:19 component_id 65535
2018/03/04 17:13:19 sub_component_id: 0
2018/03/04 17:13:19 path: sensor_1000:/junos/system-users-information:/junos/system-users-
information/
2018/03/04 17:13:19 sequence_number: 16689
2018/03/04 17:13:19 timestamp: 1520183589391
2018/03/04 17:13:19 sync_response: %!d(bool=false)
2018/03/04 17:13:19 key: __timestamp__
2018/03/04 17:13:19 uint_value: 1520183589391
2018/03/04 17:13:19 key: __junos_re_stream_creation_timestamp__
2018/03/04 17:13:19 uint value: 1520183589372
2018/03/04 17:13:19 key: __junos_re_payload-get_timestamp__
2018/03/04 17:13:19 uint_value: 1520183589390
2018/03/04 17:13:19 key: /junos/system-users-information/uptime-information/date-time
2018/03/04 17:13:19 str-value: 5:13PM
2018/03/04 17:13:19 key: /junos/system-users-inforamtion/uptime-information/up-time
2018/03/04 17:13:19 str-value: 1 day, 4:10
2018/03/04 17:13:19 key: /junos/system-users-information/uptime-information/active-user-count
2018/03/04 17:13:19 int_value: 2
2018/03/04 17:13:19 key: /junos/system-users-inforamtion/uptime-information/load-average-1
2018/03/04 17:13:19 str_value: 0.62
2018/03/04 17:13:19 key: /junos/system-users-information/uptime-information/load-average-5
2018/03/04 17:13:19 str_value: 0.56
2018/03/04 17:13:19 key: /junos/system-users-inforamtion/uptime-information/load-average-15
2018/03/04 17:13:19 str_value: 0.53
2018/03/04 17:13:19 key: __prefix__
2018/03/04 17:13:19 str_value: /junos/system-users-information/uptime-information/user-table/
user-entry[user='ab']/
2018/03/04 17:13:19 key: tty
2018/03/04 17:13:19 str_value: pts/1
2018/03/04 17:13:19 key: from
2018/03/04 17:13:19 str-value: 172.16.04.25
2018/03/04 17:13:19 key: login-time
2018/03/04 17:13:19 str_value: 5:12PM
2018/03/04 17:13:19 key: idle-time
2018/03/04 17:13:19 str-value: -
2018/03/04 17:13:19 key: command
2018/03/04 17:13:19 str_value: -cl
2018/03/04 17:13:19 system_id: vmxdockerlight_vmx1_1
2018/03/04 17:13:19 component_id: 65535

```

```
2018/03/04 17:13:19 sub_component_id: 0
2018/03/04 17:13:19 <output truncated>
```

The sample query shown below shows two sensor reports per path, then I terminated it with Ctrl-C. Every key is sent in human readable form as an absolute path and in case of lists, contains an index in form of XPATH, ideal to group values from a (time series) database like InfluxDB e.g. `/junos/system-users-information/uptime-information/user-table/user-entry[user='ab']/`

3. Verify that the module (sensor) is loaded using the `show system yang package sysusers` command, where `sysusers` is the name of the package:

```
user@switch> show system yang package sysusers
Package ID           :sysusers
XML Proxy YANG Module(s) :xmlproxyd_sysusers.yang
```

4. Enable gRPC in the Junos OS configuration:

```
user@switch> set system services extension-service request-response grpc port 32767
```

Installing a User-Defined YANG File

To add, validate, modify, or delete a user-defined YANG file for XML proxy for the Junos telemetry interface, use the `request system yang` set of commands from the operational mode:

1. Specify the name of the XML proxy YANG file and the file path to install it. This command creates a `.json` file in the `/opt/lib/render` directory.

```
user@switch> request system yang add package package-name proxy-xml module file-path-name
```



NOTE: This command can be performed only on the current routing engine.

To add multiple YANG modules with the `request system yang add package package-name proxy-xml module` command, enclose the *file-path-name* in brackets: `[file-path-name 1 file-path-name 2]`

2. (Optional) Validate an module before adding it to the router using the **request system yang validate proxy-xml module *module-name*** command. .

```
user@switch> request system yang validate proxy-xml module module-name
```

The output XML proxy YANG module validation for `xmlproxyd_<module-name>` : SUCCESS indicates successful module validation.

Mismatch error sometimes occur. If the command returns the error below, you can eliminate the error by using Junos OS Release 17.3R2 or later:

```
user@switch> request system yang validate proxy-xml module xmlproxyd_sysusers.yang
error: illegal identifier <identifier> , must not start with [xX][mM][lL]
```

3. (Optional) Update an existing XML proxy YANG file that was previously added.

```
user@switch> request system yang update package-name proxy-xml module file-path-name
```

4. Delete an existing XML proxy YANG file.

```
user@switch> request system yang delete package-name
```

5. Verify that the YANG file has been installed by entering the `show system yang package` command.

```
user@switch> show system yang package package-name
```

Troubleshoot Telemetry Sensors

IN THIS SECTION

- [Problem | 150](#)

Problem

Description

Use the following methods to troubleshoot user-define telemetry sensors:

- Execute a tcpdump for the interface your gRPC requests came from (for this task, interface fxp0 was used).

```
user@switch>monitor traffic interface fxp0 no-resolve matching "tcp port 32767"
```

- Enable traceoptions using the **set services analytics traceoptions flag xmlproxy** command. Check the xmlproxycd log file for confirmation of whether the CLI command's RPC was sent and if a response was received:

1. Issue the **show log xmlproxycd** command to show the xmlproxycd log. The value for the field xmlproxy_execute_cli_command: indicates if the RPC was sent or not. The value for the field xmlproxy_build_context indicates the command.

```
user@switch>show log xmlproxycd
Mar 4 18:52:46 vmxdockerlight_vmx1_1 clear-log[52495]: logfile cleared
Mar 4 18:52:51 xmlproxy_telemetry_start_streaming: sensor /junos/system-users-information/
Mar 4 18:52:51 xmlproxy_build_context: command show system users merge-tag:
Mar 4 18:52:51 <command format="xml">show system users</command>
Mar 4 18:52:51 xmlproxy_execute_cli_command: Sent RPC..
Mar 4 18:52:51 <system-users-information xmlns="http://xml.juniper.net/junos/17.4R1/junos"
xmlns:junos="http://xml.juniper.net/junos/*/junos">
  <uptime-information>
    <date-time junos:seconds="1520189571">
      6:52PM
    </date-time>
    <up-time junos:seconds="107400">
      1 day, 5:50
    </up-time>
    <active-user-count junos:format="1 users">
      1
    </active-user-count>
    <load-average-1>
      0.94
    </load-average-1>
    <load-average-5>
```

```
0.73
</load-average-5>
<load-average-15>
0.65
```

RELATED DOCUMENTATION

Understanding YANG on Devices Running Junos OS

[Guidelines for gRPC and gNMI Sensors \(Junos Telemetry Interface\)](#)

Send Requests to the NETCONF Server

Support for Zero Suppression

The zero-suppression feature on Juniper devices is used to manage how data is exported from sensors. When zero-suppression is enabled, any sensor values that are zero are not sent to the collector. This helps reduce the amount of data being transmitted and stored, focusing only on non-zero values.

However, if zero-suppression is disabled (using the [no-zero-suppression](#) configuration), all sensor values, including zeros, are sent to the collector. This ensures that you see all counters, even those with a value of zero.

Starting with Junos release 23.4R1, all zeros are streamed for MPLS LSP statistics even if no-zero-suppression is configured.

Configure SR-TE for Uncolored Ingress LSPs

IN THIS SECTION

- [Supporting Uncolored Sensor Paths | 152](#)
- [Configure SR-TE for Uncolored SR-TE Tunnel Statistics | 153](#)
- [Configure SR-TE for Uncolored SR-TE Per-tunnel Statistics | 153](#)

Enable SR-TE telemetry statistics by first configuring SR-TE on a device.

Supporting Uncolored Sensor Paths

Segment routing uses a segment list to enable static segment routing LSPs to steer traffic based on segment routing policies. When a segment list is used by a protocol, the protocol validates these segment identifiers and selects valid segments for traffic engineering.

When configured, the `segment-list` of a segment routing traffic engineering (SR-TE) LSP accepts both IP addresses and labels. There are two models based on these options:

1. Direct resolution- In this model, LSP accepts the IP address as the first hop.
2. Indirect resolution- This model uses only labels.

Table 9 on page 152 shows the segment routing traffic engineering (SR-TE) ingress sensors that would be installed on the packet forwarding engine in order to view the statistics that are being enabled in an SR-TE configuration.

Table 9: SR-TE Ingress Sensors

Path	Sensor	Sensor Name	Sensor Subtype	Description
2	<code>/junos/services/segment-routing/traffic-engineering/tunnel/ingress/usage/</code>	ingress-lsp1	SRTE TUNNEL INGRESS	Streams uncolored SR-TE tunnel statistics.
3	<code>/junos/services/segment-routing/traffic-engineering/tunnel/lsp/ingress/usage/</code>	i;st;0;f;lsp 1;path1 i;st;0;f;lsp 1;path2	SRTE PER- TUNNEL PER-LSP INGRESS	Streams uncolored SR-TE per-tunnel statistics.

In order to stream statistics for these sensors, you must configure SR-TE for each path and enable statistics in order to collect data that can be streamed through a sensor subscription from a device to a collector. See the sections below for sample configurations and operational mode output.

Configure SR-TE for Uncolored SR-TE Tunnel Statistics

To stream uncolored SR-TE tunnel statistics, refer to the uncolored SR-TE configuration below to create a segment list and enable static segment routing LSPs to steer traffic based on segment routing policies.

Configure Path 2

```
[edit]
set protocols source-packet-routing segment-list path1 hop1 label 50100
set protocols source-packet-routing segment-list path2 hop1 label 50100
set protocols source-packet-routing source-routing-path lsp1 to 100.100.100.100
set protocols source-packet-routing source-routing-path lsp1 primary path1
set protocols source-packet-routing source-routing-path lsp1 primary path2
set protocols source-packet-routing telemetry statistics
```

Use the `show agent sensors detail operational mode` command to display ingress-lsp1 sensor details.

```
user@host> show agent sensors detail
```

SID	Name	Type	Subtype	Packets
Bytes	Requests	Replies	Count	CounterTokens[updates]
281478734807051	ingress-lsp1	SR	srte-uncolored-ingress	10
920	12	12	1	1615[36],

Configure SR-TE for Uncolored SR-TE Per-tunnel Statistics

To stream uncolored SR-TE per-tunnel statistics, refer to the uncolored SR-TE configuration below to create a segment list and enable static segment routing LSPs to steer traffic based on segment routing policies.

Configure Path 3

```
[edit]
set protocols source-packet-routing segment-list path1 hop1 label 50100
set protocols source-packet-routing segment-list path2 hop1 label 50100
set protocols source-packet-routing source-routing-path lsp1 to 100.100.100.100
set protocols source-packet-routing source-routing-path lsp1 primary path1
```

```
set protocols source-packet-routing source-routing-path lsp1 primary path2
set protocols source-packet-routing telemetry statistics per-source per-segment-list
```

Use the `show agent sensors detail operational mode` command to display sensor `i;st;0;f;lsp1;path1` and sensor `i;st;0;f;lsp1;path2` details.

```
user@host> show agent sensors detail
```

SID	Name	Type	Subtype	Packets
Bytes	Requests	Replies	Count	CounterTokens[updates]
281478734807049	i;st;0;f;lsp1;path1	SR	srte-per-lsp-ingress	6
552	24	24	1	1611[18514],
281478734807050	i;st;0;f;lsp1;path2	SR	srte-per-lsp-ingress	0
0	24	0	1	1615[36],

SEE ALSO

| *routing-instance (Junos Telemetry Interface)*

Per-Path or Per-Segment-List Traffic Statistics

IN THIS SECTION

- [Configuring Per-Path Traffic Statistics | 155](#)

The Segment Routing Traffic Engineering (SRTE) Traffic Statistics Collection feature enables you to gather traffic statistics for both ingress IP traffic and transit MPLS traffic traversing SRTE tunnels. This functionality allows you to configure statistics at both the tunnel level and the per-path level, providing granular insights into traffic flow. This section describes about the applied per-path or per-segment-list traffic statistics that is unique for each sub candidate path of a SRTE tunnel.

You can configure per-segment-list to enable SRTE to generate a sensor ID and statistics handler. The statistics handler, including the sensor ID, is downloaded as part of SRTE route gateways. It is used in the gateway during route programming, pushing sensor information to the Packet Forwarding Engine

(PFE) if the route is downloaded to the Forwarding Information Base (FIB). To collect traffic statistics from the PFE, two optional path-level sensors are available: Ingress and Transit.

Sensor name generation varies for colored and uncolored tunnels.

- For colored tunnels, the sensor name format remains unchanged. The segment ID, previously filled with the sub-candidate path ID, is now filled with the segment-list ID.
- For uncolored tunnels, reuse the last component of the sensor name (lsp-name) to fill the segment-list ID instead of the sub-candidate path name. The segment ID is filled in hexadecimal format for both tunnel types.

The sample for sensor name format and the change in sensor name format are highlighted.

Table 10: Sensor Name Format

Tunnel Type	Sensor Type	Current Format	Format after the change
Colored	Ingress	4.4.4.4-2-1e-0-0.0.0.0-0-80	4.4.4.4-2-1e-0-0.0.0.0-0-80
	Transit	4.4.4.4-2-1e-0-0.0.0.0-0-80-f4241	4.4.4.4-2-1e-0-0.0.0.0-0-80-f4241
Uncolored	Ingress	i;st;0;f;tunnel-1; SCpath1	i;st;0;f;tunnel-1; 80
	Transit	t;st;0;f;tunnel-1; SCpath1	t;st;0;f;tunnel-1; 80

Configuring Per-Path Traffic Statistics

IN THIS SECTION

- [Configuration](#) | 156

Configuration

By configuring per-segment-list sensor, a unique telemetry sensor is created for each segment-list.

```

user@host# show protocols source-packet-routing telemetry
statistics {
    per-source {
        per-segment-list;
    }
}

user@host# show protocols source-packet-routing segment-list SCpath2
hop1 label 400300;
hop2 label 400400;

{master}[edit]
user@host# show protocols source-packet-routing segment-list SCpath1
hop1 label 400400;
user@host# show protocols source-packet-routing source-routing-path tunnel-1
to 4.4.4.4;
binding-sid 1000010;
primary {
    SCpath1;
    SCpath2;
}

user@host# show spring-traffic-engineering lsp name tunnel-1 detail

```

The output of the above configuration is as shown here:

```

Name: tunnel-1
Tunnel-source: Static configuration
Tunnel Forward Type: SRMPLS
To: 4.4.4.4-10<c>
Binding SID Label:1000010
    20 Bit MPLS Label, State: Allocated
State: Up
    Path: SCpath1
    Path Status: Up
    Outgoing interface: NA
    Auto-translate status: Disabled Auto-translate result: N/A
    Compute Status:Enabled , Compute Result:succcess , Compute-Profile Name:pro1
    Total number of computed paths: 2

```

```

Segment ID : 128
Telemetry Ingress statistics:
Sensor-name: 4.4.4.4-a-1e-0-0.0.0-0-80, Id: 3758096417
Telemetry transit statistics:
Sensor-name: 4.4.4.4-a-1e-0-0.0.0-0-80-f424a, Id: 3758096418
Computed-path-index: 1
  BFD status: N/A BFD name: N/A
  BFD remote-discriminator: N/A
  TE metric: 30, IGP metric: 30
  Delay metrics: Min: 50331645, Max: 50331645, Avg: 50331645
  Metric optimized by type: TE
  computed segments count: 3
    computed segment : 1 (computed-adjacency-segment):
      label: 400103
      source router-id: 1.1.1.1, destination router-id: 2.2.2.2
      source interface-address: 11.2.1.1, destination interface-address: 11.2.1.2
    computed segment : 2 (computed-adjacency-segment):
      label: 400202
      source router-id: 2.2.2.2, destination router-id: 3.3.3.3
      source interface-address: 22.1.1.2, destination interface-address: 22.1.1.3
    computed segment : 3 (computed-adjacency-segment):
      label: 400301
      source router-id: 3.3.3.3, destination router-id: 4.4.4.4
      source interface-address: 33.1.1.3, destination interface-address: 33.1.1.4
Segment ID : 129
Telemetry Ingress statistics:
Sensor-name: 4.4.4.4-a-1e-0-0.0.0-0-81, Id: 3758096419
Telemetry transit statistics:
Sensor-name: 4.4.4.4-a-1e-0-0.0.0-0-81-f424a, Id: 3758096420
Computed-path-index: 2
  BFD status: N/A BFD name: N/A
  BFD remote-discriminator: N/A
  TE metric: 30, IGP metric: 30
  Delay metrics: Min: 50331645, Max: 50331645, Avg: 50331645
  Metric optimized by type: TE
  computed segments count: 3
    computed segment : 1 (computed-adjacency-segment):
      label: 400105
      source router-id: 1.1.1.1, destination router-id: 2.2.2.2
      source interface-address: 11.4.1.1, destination interface-address: 11.4.1.2
    computed segment : 2 (computed-adjacency-segment):
      label: 400202
      source router-id: 2.2.2.2, destination router-id: 3.3.3.3

```

```

        source interface-address: 22.1.1.2, destination interface-address: 22.1.1.3
    computed segment : 3 (computed-adjacency-segment):
        label: 400301
        source router-id: 3.3.3.3, destination router-id: 4.4.4.4
        source interface-address: 33.1.1.3, destination interface-address: 33.1.1.4
    Path: SCpath2
    Path Status: Up
    Outgoing interface: NA
    Auto-translate status: Disabled Auto-translate result: N/A
    Compute Status:Enabled , Compute Result:success , Compute-Profile Name:pro2
    Total number of computed paths: 2
    Segment ID : 256
    Telemetry Ingress statistics:
    Sensor-name: 4.4.4.4-a-1e-0-0.0.0.0-0-100, Id: 3758096421
    Telemetry transit statistics:
    Sensor-name: 4.4.4.4-a-1e-0-0.0.0.0-0-100-f424a, Id: 3758096422
    Computed-path-index: 1
    BFD status: N/A BFD name: N/A
    BFD remote-discriminator: N/A
    TE metric: 30, IGP metric: 30
    Delay metrics: Min: 50331645, Max: 50331645, Avg: 50331645
    Metric optimized by type: Minimum-Delay
    computed segments count: 3
        computed segment : 1 (computed-adjacency-segment):
            label: 400103
            source router-id: 1.1.1.1, destination router-id: 2.2.2.2
            source interface-address: 11.2.1.1, destination interface-address: 11.2.1.2
        computed segment : 2 (computed-adjacency-segment):
            label: 400202
            source router-id: 2.2.2.2, destination router-id: 3.3.3.3
            source interface-address: 22.1.1.2, destination interface-address: 22.1.1.3
        computed segment : 3 (computed-adjacency-segment):
            label: 400301
            source router-id: 3.3.3.3, destination router-id: 4.4.4.4
            source interface-address: 33.1.1.3, destination interface-address: 33.1.1.4
    Segment ID : 257
    Telemetry Ingress statistics:
    Sensor-name: 4.4.4.4-a-1e-0-0.0.0.0-0-101, Id: 3758096423
    Telemetry transit statistics:
    Sensor-name: 4.4.4.4-a-1e-0-0.0.0.0-0-101-f424a, Id: 3758096424
    Computed-path-index: 2
    BFD status: N/A BFD name: N/A
    BFD remote-discriminator: N/A

```

```

TE metric: 30, IGP metric: 30
Delay metrics: Min: 50331645, Max: 50331645, Avg: 50331645
Metric optimized by type: Minimum-Delay
computed segments count: 3
  computed segment : 1 (computed-adjacency-segment):
    label: 400105
    source router-id: 1.1.1.1, destination router-id: 2.2.2.2
    source interface-address: 11.4.1.1, destination interface-address: 11.4.1.2
  computed segment : 2 (computed-adjacency-segment):
    label: 400202
    source router-id: 2.2.2.2, destination router-id: 3.3.3.3
    source interface-address: 22.1.1.2, destination interface-address: 22.1.1.3
  computed segment : 3 (computed-adjacency-segment):
    label: 400301
    source router-id: 3.3.3.3, destination router-id: 4.4.4.4
    source interface-address: 33.1.1.3, destination interface-address: 33.1.1.4

```

SRv6 and SRv6-TE Traffic Sensor Telemetry

SUMMARY

Segment Routing over IPv6 (SRv6) telemetry support introduces a powerful mechanism for monitoring and optimizing network performance. By leveraging the native SRv6 data model, you can integrate telemetry tools that provide comprehensive insights into network traffic. These insights are gathered using streaming XPath from the SRv6 Manager and rpd, which collect and transmit detailed traffic statistics. This capability is essential for maintaining high levels of network performance and security, as it allows you to monitor real-time traffic patterns, detect anomalies, and respond to potential issues promptly.

IN THIS SECTION

- [Benefits of SRv6 and SRv6-TE Traffic Sensor Telemetry | 162](#)
- [Limitations | 163](#)

The telemetry support for SRv6 includes the integration of a unified data model for both SRv6 and SRv6 TE tunnels. This standardization simplifies the management and monitoring of different tunneling technologies within your network. Although the current implementation focuses on SRv6 TE tunnels, the groundwork is laid for broader support in future updates. The data model ensures consistency and ease of use, making it simpler to configure and manage telemetry across various network segments.

To enable effective traffic monitoring, traffic sensors are installed on routes and next hops within the rpd. These sensors are mapped to ISIS and SRTE sensors, allowing the Packet Forwarding Engine (PFE) to collect and stream traffic statistics. This integration provides you with detailed visibility into network performance, enabling proactive management and troubleshooting. By monitoring traffic at a granular level, you can optimize routing decisions, improve network efficiency, and enhance overall operational reliability.

Following are some of the functionalities of SRv6 in telemetry:

- **Traffic Statistics telemetry for IGP (ISIS Only) Routes / NextHops:** Following is the list of PFE sensors, which are installed by IS-IS based upon the newly introduced configuration commands and streamed by PFE.

Table 11: Traffic Statistics telemetry for IGP Routes or NextHops.

Sensor Path	Description
/state/routing-instances/routing-instance[name]/ protocols/source-packet-routing/srv6/remote- locators/remote-locator[address]/counters/in-pkts/ state/routing-instances/routing-instance[name]/ protocols/source-packet-routing/srv6/remote- locators/remote-locator[address]/counters/in-octets	This sensor path is applicable for PFE sensor for Remote SRv6 locator routes in INET6.0 RT.
/state/routing-instances/routing-instance[name]/ protocols/source-packet-routing/srv6/remote- locators/remote-locator[address]/counters/out-pkts/ state/routing-instances/routing-instance[name]/ protocols/source-packet-routing/srv6/remote- locators/remote-locator[address]/counters/out-octets	This sensor path is applicable for PFE sensor for Remote SRv6 Locator Route TCNH Gateway in INET6.3 RT.

- **Support for Uniformed Native Data Model for SR-TE:** This feature includes a model tree for both colored and uncolored policies. Only SRv6-TE FIB Xpaths (traffic counter) with ingress counters are supported for both tunnel types. SRv6 TE traffic counter Xpaths streaming is platform-dependent, applicable only to JUNOS MX platforms. At PFE, stats accounting occurs before policy application, supporting only ingress counters. SRv6-TE sensors support only the DEFAULT instance.
- **Colored SRv6-TE per policy ingress counters:** The supported counters are listed here-

Table 12: Colored SRv6-TE per-policy Ingress Counters

Sensor Path	Description
/state/routing-instances/routing-instance[name]/protocols/source-packet-routing/te-policies/te-policy[color endpoint]/counters/srv6/in-bytes	This sensor path is applicable for per-policy level PFE sensors pfor SRv6 SR-TE route TCNH gateway in junos-rti table.
/state routing-instances/routing-instance[name]/protocols/source-packet-routing/te-policies/te-policy[color endpoint]/counters/srv6/in-pkts	This sensor path is applicable for ingress SRv6 traffic.

Note that when per-policy counters are streamed, the values will be cumulative of all the candidate-paths of that policy.

- **Colored SRv6-TE per path ingress counters:** The supported counters are listed here-

Table 13: Colored SRv6-TE per-path Ingress Counters

Sensor Path	Description
/state/routing-instances/routing-instance[name]/protocols/source-packet-routing/te-policies/te-policy[color endpoint]/candidate-paths/candidate-path[protocol-origin originator-asn originator-address discriminator]/segment-lists/segment-list[segment-id]/counters/in-bytes	This sensor path is applicable for per-path level PFE sensors for SRv6 SR-TE route TCNH gateway in junos-rti table.
/state/routing-instances/routing-instance[name]/protocols/source-packet-routing/te-policies/te-policy[color endpoint]/candidate-paths/candidate-path[protocol-origin originator-asn originator-address discriminator]/segment-lists/segment-list[segment-id]/counters/in pkts	This sensor path is applicable for ingress SRv6 traffic.

- **Uncolored SRv6-TE per tunnel ingress counters:** The supported counters are listed here-

Table 14: Uncolored SRv6-TE per-tunnel Ingress Counters

Sensor Path	Description
/state/routing-instances/routing-instance[name]/protocols/source-packet-routing/uncolored-te-tunnels/uncolored-te-tunnel[name]/counters/in-bytes	This sensor path is applicable for per-policy level PFE sensors for SRv6 SRTE uncolored route TCNH gateway in inet6.3 table.
/state/routing-instances/routing-instance[name]/protocols/source-packet-routing/uncolored-te-tunnels/uncolored-te-tunnel[name]/counters/in-pkts	This sensor path is applicable for ingress SRv6 traffic.

- **Uncolored SRv6-TE per path ingress counters:** The supported counters are listed here-

Table 15: Uncolored SRv6-TE per-path Ingress Counters

Sensor Path	Description
/state/routing-instances/routing-instance[name]/protocols/source-packet-routing/uncolored-te-tunnels/uncolored-te-tunnel[name]/segment-lists/segment-list[segment-id]/counters/in-bytes	This sensor path is applicable for per-path level PFE sensors for SRv6 SRTE uncolored route TCNH gateway in inet6.3 table.
/state/routing-instances/routing-instance[name]/protocols/source-packet-routing/uncolored-te-tunnels/uncolored-te-tunnel[name]/segment-lists/segment-list[segment-id]/counters/in-pkts	This sensor path is applicable for ingress SRv6 traffic.

Benefits of SRv6 and SRv6-TE Traffic Sensor Telemetry

- Enhances network performance monitoring through detailed traffic statistics, allowing you to track packet and byte counts for both ingress and egress traffic.
- Provides granular insights into SRv6 traffic via traffic sensors for IS-IS SRv6 routes and next-hops, facilitating effective network analysis and optimization.
- Supports targeted troubleshooting and optimization by offering ingress counters for both colored and uncolored SRv6-TE policies and paths, enabling precise traffic performance analysis.

- Improves operational efficiency and decision-making through the integration of native YANG data models for SRv6 Base and SRv6-TE, allowing for seamless telemetry data streaming and interpretation.

Limitations

- SR-TE recommends using unique tunnel names for different sources such as Static, PCEP, or BGP-SRTE. Sharing names will result in shared stats.
- Sensor values do not retain during the NSR/GR phase. They change after NSR/GR.

On-Box Aggregation Overview

IN THIS SECTION

- [Aggregation Logic | 164](#)
- [Packet Forwarding Engine Data Export | 165](#)
- [Identifying Aggregated Data Samples | 166](#)
- [Usage Tips | 167](#)
- [Supported Resource Paths | 167](#)

Juniper devices support on-box aggregation. Interface, CoS, MPLS, AF, and AE counters are aggregated on a Juniper device, then the telemetry data is streamed to a collector. Compared to off-box aggregation, where counters are streamed to the collector and aggregated there, on-box aggregation presents more accurate telemetry data that recognizes systemic events, such as line card resets or LAG membership changes. On-box aggregation reduces production errors at the collector. It also significantly reduces the overall telemetry data bandwidth utilization in certain cloud-based environments in which data is metered.

Aggregation Logic

Data aggregation is the simple addition of given counter values from all the Packet Forwarding Engine instances received as Google protobuf messages in the na-grpcd daemon. The aggregation module in na-grpcd daemon caches statistics from multiple Packet Forwarding Engines based on the subscription request received. Caching is done per subscription received.

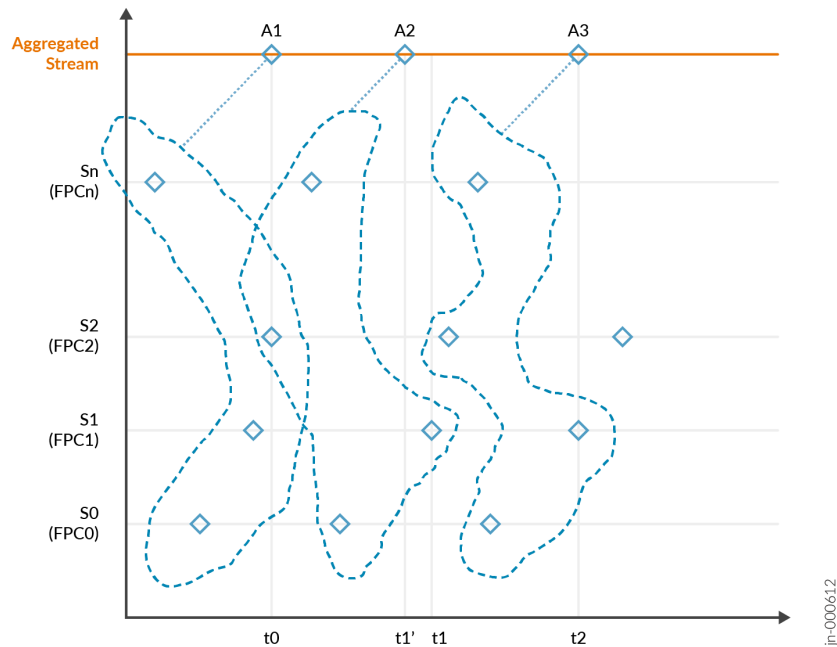
The following points apply to data aggregation:

- If statistical data is gauge type, field values will be either Average, Max or Min, and the aggregation module orders aggregation based on the gauge type field value.
- Aggregation is done for both counter and gauge data types annotated in the YANG file.
- At any time, there is only one source of data for a given Packet Forwarding Engine available in the cache. New entries overwrite the existing cache.
- No separate timer aggregates or streams data. Whenever sample data is received from a Packet Forwarding Engine, the time to aggregate and stream is checked with the previously sent aggregated timestamp. For example:
 - $\text{current time stamp} - \text{previously aggregated time stamp} > \text{reporting_interval}$
- The export timestamp carries a median timestamp for the aggregated data.
- Record caching in the na-grpcd daemon is available only while the subscription is valid (active). When a subscription is inactive, the exported aggregated telemetry data might not match the Routing Engine command line interface statistics. The following triggers may create this mismatch:
 - Restart of one or more FPCs in which the child members are part of the aggregated interface.
 - Deletion of up to n-1 child members (at least one child link is active) from an aggregate interface.
- Lastly, if the subscription is deleted, the associated cache also is deleted.

In a gNMI or gRPC stream, the first sample may not contain aggregated statistics counter values because the first interval from the Routing Engine may have only received one sample from multiple FPCs. However, from the second interval onwards, the stream will contain aggregated counter values received from all the FPCs.

Aggregation of samples is triggered based upon one of the incoming samples. Shown in [Figure 6 on page 165](#), sample S2 is delayed. Because of this delay, A2 aggregation uses the old t0 sample with a median timestamp of the samples. This is labeled t1' (though actually it is t1).

Figure 6: Aggregation of Sample Intervals



Packet Forwarding Engine Data Export

The aggregation feature relies on the fundamental protobuf-based telemetry interface to export data from the Packet Forwarding Engine. A protobuf-based telemetry stream encapsulates data for export in a common, top-level envelope called **TelemetryStream** (shown below).

```
message TelemetryStream {
  // router hostname
  // (or, just in the case of legacy (microkernel) PFEs, the 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 increasing 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];
...
}

```

This top-level envelope provides necessary information to build the Packet Forwarding Engine statistics cache in an operation called *combinator*. The combinator performs a logical join on multiple data instances that have a common interface ID and queue ID, and produces system-wide aggregated output queue statistics. Every incoming protobuf is organized into the cache using sensor, component and a Packet Forwarding Engine slot field in the **TelemetryStream** protobuf. The combinator keeps track of instance data per Packet Forwarding Engine by caching the latest protobuf export from the Packet Forwarding Engine for a specific sensor. Using binary protobuf as an aggregation unit ensures minimal caching and bookkeeping overhead for each collection of counters contained within the protobuf.

A timestamp on the telemetry data exported from the Packet Forward Engine triggers the aggregate data computation. If the elapsed time since the previous aggregate data export exceeds the sensor refresh interval, then a new aggregate value is exported. This new value uses the timestamp from the last Packet Forwarding Engine sample that triggered the aggregated value export. This method eliminates the need to run a per-sensor timer inside the combinator to reap aggregated data. Also, caching the last received data export from the Packet Forwarding Engine minimizes "drift" in the exported aggregate data. This is especially useful for instances such as a delayed Packet Forwarding Engine export as well as operational changes that can potentially preclude further export from the Packet Forwarding Engine.

Identifying Aggregated Data Samples

You can identify aggregated samples by the header field component with the value `na-grpcd`. The `na-grpcd` daemon produces the aggregated data.

```

system_id: r1q13dep
component_id: 65535
sensor_name: sensor_1004_1_1
subscribed_path: /interfaces/interface/subinterfaces/subinterface/
streamed_path: /interfaces/interface/subinterfaces/subinterface/

```

```

component: na-grpcd
sequence_number: 2097152
export_timestamp: 1663783826926
update {
  timestamp: 1663783826901000000
  prefix: /interfaces/interface[name='ge-1/0/0']/subinterfaces/subinterface[index='16386']
update {
  path {
    elem {
      name:  init-time
    }
  }
  val {
    uint_val: 1663780730
  }
}
}

```

Usage Tips

Keep the following considerations in mind when using on-box aggregation:

- The packet and byte rates calculation is based on the aggregated statistics timestamp message. Since the timestamp message is synthesized on one Routing Engine, but is based on cached data from multiple FPCs, the packet and byte rate may not be accurate. Accuracy is higher when the reporting interval configuration is reasonable (as per best practices guidelines).
- If statistics messages are not exported by the Packet Forwarding Engines (for example, the operational status is down), then aggregation statistics are not exported to the collector.
- Expect telemetry MPLS LSP statistics with an AE bundle to have a 5% deviation in packet and byte rates.

Supported Resource Paths

On-box aggregations supports these resource paths:

- `/junos/system/linecard/interface/traffic/`
- `/junos/system/linecard/interface/queue/`

- `/junos/system/linecard/interface/logical/usage/`
- `/junos/system/linecard/cos/interface/interface-set/output/queue/`
- `/junos/services/label-switched-path/usage/`
- `/qos/interfaces/interface/output/queues/queue/state/`
- `/interfaces/interface/state/counters/`
- `/interfaces/interface/subinterfaces/subinterface/state/counters/`
- `/interfaces/interface/subinterfaces/subinterface/ipv4/state/counters/`
- `/interfaces/interface/subinterfaces/subinterface/ipv6/state/counters/`
- `/network-instances/network-instance/mpls/lsp/constrained-path/tunnels/tunnel/state/counters/`
- `/junos/system/linecard/interface/queue/` (CoS native resource path)
- `/junos/system/linecard/qmon-sw/` (CoS native resource path)
- `/qos/interfaces/interface/output/queues/queue/state/` (CoS OpenConfig resource path)
- `/qos/interfaces/interface/input/virtual-output-queues/voq-interface/queues/queue/state/` (CoS OpenConfig resource path)
- `/junos/system/linecard/interface/queue/` (native path for physical interface statistics, AE interfaces supported)
- `/qos/interfaces/interface/output/queues/queue/state/` (AE interfaces supported)

See [Junos YANG Data Model Explorer](#) for more information about resource paths, their supported leaves, and the device platforms that support them.

Change History Table

Feature support is determined by the platform and release you are using. Use [Feature Explorer](#) to determine if a feature is supported on your platform.

Release	Description
23.2R1	Starting in Junos OS Release 23.2R1, on-box aggregation is supported on interface, CoS, MPLS, and aggregated Ethernet counters. Supported platforms include MX150, MX204, MX240, MX304, MX480, MX960, MX2008, MX2010, MX2020, MX10003, MX10004, MX10008, MX10016, and vMX.

Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets

IN THIS SECTION

- [Understanding Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets | 169](#)
- [Enable Export of Subscriber Statistics and Queue Statistics | 171](#)
- [Guidelines for Exporting Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets | 173](#)
- [gRPC Sensors for Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets \(Junos Telemetry Interface\) | 174](#)

Understanding Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets

IN THIS SECTION

- [About Subscriber and Queue Statistics | 169](#)
- [Enabling Export of Statistics | 170](#)

You can use subscriber statistics and queue statistics for dynamic interfaces and interface-sets to support remote analytics and monitor Juniper devices that operate as a Broadband Network Gateway (BNG). Using these statistics, you can model and condition traffic flows in a subscriber access network.

About Subscriber and Queue Statistics

Subscriber statistics include the per-IP protocol family (IPv4 or IPv6) packet information (receive and transmitted packets and bytes) for a subscriber interface. They will only include subscriber data forwarded by the system. Filtered and dropped packets and control traffic are factored out and not delivered.

ON-CHANGE subscription support for interface meta-data sends asynchronous notifications when interfaces are created and deleted. After an initial baseline of delivering create notifications for all existing interfaces, only notifications for interfaces that are being created or deleted are sent to an external collector.

Use queue statistics to determine oversubscription levels, the mix of forwarding-class traffic, or traffic rates for a given CoS-enabled interface or interface-set.

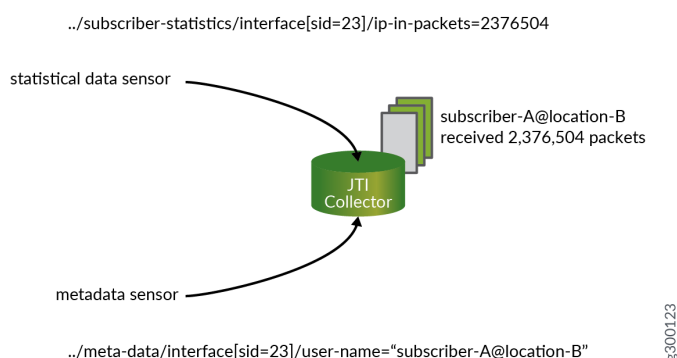
To receive subscriber statistics, you also must enable RADIUS accounting. See *802.1X and RADIUS Accounting*.

Enabling Export of Statistics

To receive statistics, you enable both meta-data and statistical data for export on your Juniper device through the Junos CLI. Meta-data for the interface is provided because the interface key is a dynamic integer, a session identifier (SID), which conveys no context to an external server. The meta-data provides more tangible context (such as the user name, a profile name VLAN tags, etc.) to the SID. An external collector associates the statistical data to a persistent reference.

A subscription for both statistical data and meta-data can be made from the external collector (in [Figure 7 on page 170](#), the JTI collector). In this way, the two streams are “merged” and a correlation is made between the statistical data and the meta-data. The dynamic SID is matched with the more permanent attributes such as user name and location.

Figure 7: JTI Collector “Merging” Sensor Data



SEE ALSO

[gRPC Sensors for Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets \(Junos Telemetry Interface\)](#) | 174

Enable Export of Subscriber Statistics and Queue Statistics

You can enable the telemetry export of subscriber statistics and queue statistics for dynamic interfaces and interface-sets. After you enable telemetry for these statistics, they are eligible for export to one or more collectors using a remote procedure call (gRPC) subscription.

Use these statistics to model and condition traffic flows in a subscriber access network and to provide subscriber statistics information (accurate accounting).

To enable the export of subscriber statistics and associated interface meta-data:

1. Enable export of interface meta-data and subscriber statistics:

```
[edit dynamic-profiles profile-name]
user@host# set telemetry subscriber-statistics
```

2. Enable the logical demultiplexing (demux) interface in a dynamic profile to export subscriber accurate statistics:

```
[edit dynamic-profiles interfaces demux0]
user@host# unit $junos-interface-unit actual-transit-statistics
```

To enable export of interface meta-data and queue statistics for dynamic interfaces:

1. Enable export of interface meta-data and interface queue statistics. Use the profile variable \$junos-interface-name.



NOTE: the profile variables \$junos-interface-name and \$junos-interface-set-name are generated from the corresponding device, unit and interface-set elements in the interfaces stanza at profile instantiation time. Using these derived variables is a

convenient way to configure telemetry behavior for the interface or interface-set without the need to mimic the specific configuration in the interfaces stanza.

```
[edit dynamic-profiles profile-name]
user@host# set telemetry queue-statistics interface $junos-interface-name
```

2. To override the default internal queue-stats collection interval of 900 seconds or the default queue export filter (all queues, 0-7), add the rate and queues statements.

```
[edit dynamic-profiles profile-name telemetry queue-statistics interface $junos-interface-name]
user@host# set rate 300
user@host# set queues "0,1,2"
```

To enable export of interface-set meta-data and queue statistics for dynamic interface-sets:

1. Enable export of interface-set meta-data and interface-set queue statistics. Use the profile variable \$junos-interface-set-name.



NOTE: the profile variables \$junos-interface-name and \$junos-interface-set-name are generated from the corresponding device, unit and interface-set elements in the interfaces stanza at profile instantiation time. Using these derived variables is a convenient way to configure telemetry behavior for the interface or interface-set without the need to mimic the specific configuration in the interfaces stanza.

```
[edit dynamic-profiles profile-name]
user@host# set telemetry queue-statistics interface-set $junos-interface-set-name
```

2. To override the default internal queue-stats collection interval of 900 seconds or the default queue export filter (all queues, 0-7), add the rate and queues statements.

```
[edit dynamic-profiles profile-name telemetry queue-statistics interface-set $junos-interface-set-name]
user@host# set rate 300
user@host# set queues "0,1,2"
```

After telemetry export is enabled, meta-data and statistics can be streamed to external collectors subscribing to the available resource paths.

Use the resource paths from ["gRPC Sensors for Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets \(Junos Telemetry Interface\)"](#) on page 174 for your gRPC subscription.

SEE ALSO

Understanding Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets

[Guidelines for Exporting Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets | 244](#)

[gRPC Sensors for Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets \(Junos Telemetry Interface\) | 174](#)

[Configure a NETCONF Proxy Telemetry Sensor in Junos | 135](#)

Guidelines for Exporting Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets

You can use subscriber statistics and queue statistics for dynamic interfaces and interface-sets to support remote analytics and monitoring on MX Series routers that operate as a Broadband Network Gateway (BNG).

Before enabling export of subscriber statistics and queue statistics for dynamic interfaces and interface-sets, consider the following limitations:

- On MX Series routers supporting the Modular Port Concentrator 2 (MPC2), a slow internal refresh cycle for queue statistics can occur. This cycle can be lengthy at full line card scale. If the subscription frequency is higher than the internal refresh cycle, exported data may appear stale across reporting intervals.
- The unified in-service software upgrade (ISSU) feature enables you to upgrade your device between two different Junos OS releases with no disruption on the control plane and with minimal disruption of traffic. Dynamic interfaces and Interface-sets created prior to ISSU and prior to Junos OS Release 18.4R1 do not support telemetry for subscriber and queue statistics.
- The subscription frequency should be larger than the time to export telemetry. If the volume of data cannot be exported before the next reporting interval, the export continues to completion and the next reporting interval will immediately start. In such instances, continuous streaming results—behavior that may not be wanted.
- Multiple sensors from the dynamic-interfaces sub-tree may be subscribed to simultaneously. As streaming of these sensors for the sub-tree is supported by a single Junos component, you should expect the time to export the sensor data for each subscription to extend.

- Juniper advises to enable export only for active queues. To do this, include the `queues` statement at the `[[edit dynamic-profiles profile-name telemetryqueue-statistics $junos-interface-name]` or `[[edit dynamic-profiles profile-name telemetry queue-statistics $junos-interface-set-name]` hierarchy level. Exporting data for active queues only reduces the amount of data to export for each reporting interval.

SEE ALSO

Understanding Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets

Enable Export of Subscriber Statistics and Queue Statistics

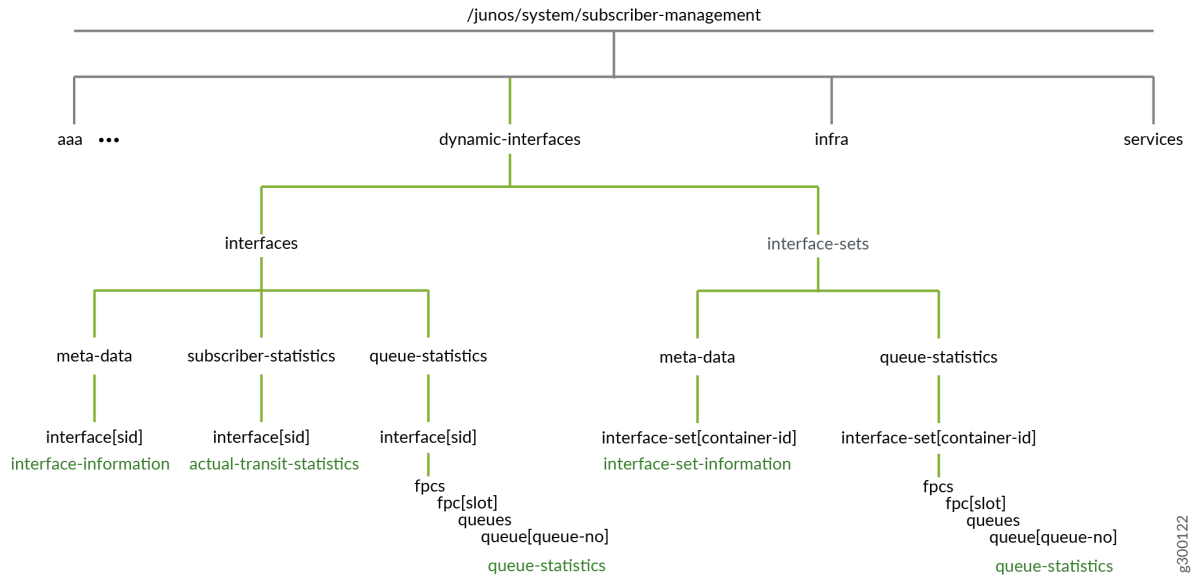
gRPC Sensors for Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets (Junos Telemetry Interface)

Starting with Junos OS Release 18.4R1, MX Series routers are supported.

You can use subscriber statistics and queue statistics for dynamic interfaces and interface-sets to support remote analytics and monitoring on Juniper devices that operate as a Broadband Network Gateway (BNG). Using these statistics, you can model and condition traffic flows in a subscriber access network.

[Figure 8 on page 175](#) shows the structure of the sensors or resource paths used for subscription to the external collector. The resource paths are a combination of both meta-data and statistical data.

Figure 8: Structure of Sensors



For statistics delivery through a gRPC subscription, include one or more resource paths from [Table 16 on page 176](#) in the subscription. For statistics delivered through gRPC, you will also need to install some additional software enable statistics to be exported on your Juniper device through the Junos CLI. For more information, see [Enable Export of Subscriber Statistics and Queue Statistics](#). For more information about creating a subscription, see ["Configure a NETCONF Proxy Telemetry Sensor in Junos"](#) on [page 135](#).

Table 16: gRPC Sensors

resource path	Description
<p>/junos/system/subscriber-management/dynamic-interfaces/interface-sets/meta-data/interface-set[container-id='container-id-value']/</p>	<p>Sensor for subscriber interface-set information.</p> <p>This sensor is supported on MX Series routers starting with Junos OS Release 18.4R1.</p> <p>ON-CHANGE streaming is supported.</p> <p>The following end paths are supported:</p> <ul style="list-style-type: none"> • cos-egress-tcp-name-The egress traffic control profile associated with this interface-set. • cos-egress-tcp-remainder-name-The egress remainder traffic control profile associated with this interface-set. • interface-set-name-The name of the interface-set as supplied by AAA or as constructed by the topology relationship (ACI string or interface stacking). • interface-set-type-The type of interface-set (determines structure of interface-set-name). • device-name-The name of the underlying device or port (e.g. ge-1/0/0 or ae1). This leaf is empty if the interface-set-type is not a physical interface-set type. • stag-The outer VLAN tag. The value is 0 if interface-set-type is not a VLAN type. • ctag-The inner VLAN tag. The value is 0 if interface-set-type is not a VLAN type.

Table 16: gRPC Sensors (*Continued*)

resource path	Description
/junos/system/subscriber-management/dynamic-interfaces/interfaces/meta-data/interface[sid='sid-value']/	<p>Sensor for subscriber interface information.</p> <p>ON-CHANGE streaming is supported.</p> <p>The following end paths are supported:</p> <ul style="list-style-type: none"> • interface-index-The system assigned interface index for the interface. • session-type-The type of client session (e.g VLAN, DHCP, PPPoE). • user-name-The login name for this interface and session. • profile-name-The name of the client profile used to create the interface. • underlying-interface-name-The name of the associated underlying interface. • cvlan-tag-The innermost VLAN tag value associated with the interface. • svlan-tag-The outermost VLAN tag value associated with the interface.

Table 16: gRPC Sensors (*Continued*)

resource path	Description
<pre>/junos/system/subscriber-management/dynamic- interfaces/interfaces/subscriber-statistics/ interface[sid='sid-value']/</pre>	<p>Sensor for actual accounting statistics for dynamic subscriber interfaces.</p> <p>The following end paths are supported:</p> <ul style="list-style-type: none"> • ip-in-packets-The number of actual transit IPv4 & IPv6 packets received by the interface. • ip-out-packets-The number of actual transit IPv4 & IPv6 packets sent to the interface. • ip-in-bytes-The number of actual transit IPv4 & IPv6 bytes received by the interface. • ip-out-bytes-The number of actual transit IPv4 & IPv6 bytes received by the interface. • ipv6-in-packets-The number of actual transit IPv6 packets received by the interface. • ipv6-out-packets-The number of actual transit IPv6 packets sent to the interface. • ipv6-in-bytes-The number of actual transit IPv6 bytes received by the interface. • ipv6-out-bytes-The number of actual transit IPv6 bytes sent to the interface.
<pre>/junos/system/subscriber-management/dynamic- interfaces/interfaces/queue-statistics/ interface[sid='sid-value']/fpcs/fpc[slot='slot- value']/queues/queue/[queue-no='queue-no-value']/</pre>	<p>Sensor for queue statistics for dynamic interfaces.</p> <p>The following end paths are supported:</p> <ul style="list-style-type: none"> • transmitted-packets-The number of actual transit IPv4 & IPv6 packets received by the interface. • transmitted-bytes-Total bytes enqueued for this queue. • dropped-packets-Total packets dropped (because of RED, rate-limited, tail-drop, etc.) for the queue. • dropped-bytes-Total bytes dropped (because of RED, rate-limited, tail-drop, and so on.) for the queue.

Table 16: gRPC Sensors (Continued)

resource path	Description
<code>/junos/system/subscriber-management/dynamic-interfaces/interface-sets/queue-statistics/interface-set[container-id='container-id-value']/fpcs/fpc[slot='slot-value']/queues/queue/[queue-no='queue-no-value']</code>	<p>Sensor for queue statistics for dynamic interface-sets.</p> <p>The following end paths are supported:</p> <ul style="list-style-type: none">transmitted-packets-The number of actual transit IPv4 & IPv6 packets received by the interface.transmitted-bytes-Total bytes enqueued for this queue.dropped-packets-Total packets dropped (because of RED, rate-limited, tail-drop, etc.) for the queue.dropped-bytes-Total bytes dropped (because of RED, rate-limited, tail-drop, etc.) for the queue.

SEE ALSO

Understanding OpenConfig and gRPC on Junos Telemetry Interface

Change History Table

Feature support is determined by the platform and release you are using. Use [Feature Explorer](#) to determine if a feature is supported on your platform.

Release	Description
18.4R1	Starting with Junos OS Release 18.4R1, MX Series routers are supported.

Enabling Export of Transit SPRING Statistics

IN THIS SECTION

- Understanding Enabling Export of Transit SPRING Statistics | 180
- Enabling Collection of SPRING Statistics and Exporting Them | 182

Understanding Enabling Export of Transit SPRING Statistics

IN THIS SECTION

- [About Transit Spring Statistics | 180](#)
- [Use a Valid SPRING Configuration | 180](#)
- [Enabling Export of Statistics | 181](#)
- [Exporting Statistics | 182](#)

Source Packet Routing in Networking (SPRING), also known as segment routing, is a control-plane architecture that enables an ingress router to steer a packet through a specific set of nodes and links in the network. Starting in Junos OS Release 19.1R1, Junos telemetry interface (JTI) supports the export of transit SPRING statistics on PTX3000 routers and PTX5000 routers with FPC2. Use these statistics to monitor traffic, model engineering, and plan capacity.

About Transit Spring Statistics

Exported statistics are for SPRING traffic and exclude RSVP and LDP-signaled traffic. Family MPLS statistics per interface is accounted for separately. The segment routing statistics also include SPRING traffic statistics per link aggregation group (LAG) member and per segment identifier (SID).

Use a Valid SPRING Configuration

Statistics are not collected without the appropriate SPRING configuration in place. Before you enable SPRING statistics, verify your SPRING configuration on the device from which you will export statistics. Below is a sample Junos OS SPRING configuration.

Ingress Device

```
user@host# set groups isis protocols isis traffic-engineering family inet shortcuts
user@host# set groups isis protocols isis source-packet-routing sensor-based-stats per-sid
ingress
user@host# set groups isis protocols isis source-packet-routing srgb start-label 400
user@host# set groups isis protocols isis source-packet-routing srgb index-range 20000
user@host# set groups isis protocols isis source-packet-routing node-segment ipv4-index 1
user@host# set groups isis protocols mpls traffic-engineering bgp-igp-both-ribs
```

```

user@host# set groups isis protocols isis traffic-engineering family inet shortcuts
user@host# set groups isis protocols isis traffic-engineering family inet-mpls shortcuts

```

Transit Device

```

user@host# set groups isis protocols isis traffic-engineering family inet shortcuts
user@host# set groups isis protocols isis source-packet-routing sensor-based-stats per-sid
ingress
user@host# set groups isis protocols isis source-packet-routing srgb start-label 400
user@host# set groups isis protocols isis source-packet-routing srgb index-range 20000
user@host# set groups isis protocols isis source-packet-routing node-segment ipv4-index 2

user@host# set groups isis system services extension-service request-response grpc clear-text
port 32767
user@host# set groups isis system services extension-service request-response grpc max-
connections 8
user@host# set groups isis system services extension-service request-response grpc skip-
authentication

```

Egress Device

```

user@host# set groups isis protocols isis traffic-engineering family inet shortcuts
user@host# set groups isis protocols isis source-packet-routing sensor-based-stats per-sid
ingress
user@host# set groups isis protocols isis source-packet-routing srgb start-label 400
user@host# set groups isis protocols isis source-packet-routing srgb index-range 20000
user@host# set groups isis protocols isis source-packet-routing node-segment ipv4-index 3

```

For more information about configuring SPRING, see *Understanding Source Packet Routing in Networking (SPRING)*.

Enabling Export of Statistics

To receive statistics, you enable statistical data for export on your Juniper device through the Junos OS. To enable export, configure **per-sid-ingress** at the **[edit protocols isis source-packet-routing]** hierarchy level. For an example of how to configure this, see ["Enabling Collection of SPRING Statistics and Exporting Them" on page 182](#).

Exporting Statistics

After you enable telemetry export for transit SPRING statistics, you can stream statistics to an outside collector using either remote procedure call (gRPC) services or a native sensor that exports statistics in UDP-format and requires a Junos OS configuration on the device.

SEE ALSO

Understanding Source Packet Routing in Networking (SPRING)

[Enabling Collection of SPRING Statistics and Exporting Them | 182](#)

Enabling Collection of SPRING Statistics and Exporting Them

You can export through Junos telemetry interface (JTI) transit Source Packet Routing in Networking (SPRING) statistics, also known as segment routing statistics.

Before statistics can be exported, they need to be enabled for collection. After statistics are enabled for collection, you choose how they should be exported to an outside collector: gRPC services or a UDP-format using a native sensor.

To enable the export of transit SPRING statistics:

Enable sensor-based statistics per segment identifier for the ingress direction:

```
[edit protocols isis source-packet-routing]
user@host# set sensor-based-stats per-sid ingress
```

To export transit SPRING statistics using gRPC services:

1. Start gRPC services for the device by including the configuration below:

```
system {
  services {
    extension-service {
      request-response {
        grpc {
          clear-text {
```

```
        port 32767;
      }
      skip-authentication;
    }
  }
}
```

- 2. Create a subscription using the `telemetrySubscribe` remote procedure call. The subscription must contain telemetry parameters, including the resource path `/junos/services/segment-routing/sid/usage/`. For more information about configuring gRPC streaming and creating a subscription for statistics to be delivered to an outside collector, see [Understanding OpenConfig and gRPC on Junos Telemetry Interface](#), [gRPC Services for Junos Telemetry Interface](#), and *Configure a NETCONF Proxy Telemetry Sensor in Junos*.

To export transit SPRING statistics in UDP-format using a native sensor:

- 1. Configure the `sensor` statement at the `[edit services analytics]` hierarchy level on the device. Include the sensor path (`/junos/services/segment-routing/sid/usage/`), 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 the data.

For more information about configuring a native sensor, see [Configuring a Junos Telemetry Interface Sensor \(CLI Procedure\)](#).

SEE ALSO

| [Understanding Source Packet Routing in Networking \(SPRING\)](#)

Change History Table

Feature support is determined by the platform and release you are using. Use [Feature Explorer](#) to determine if a feature is supported on your platform.

Release	Description
19.1R1	Starting in Junos OS Release 19.1R1, Junos telemetry interface (JTI) supports the export of transit SPRING statistics on PTX3000 routers and PTX5000 routers with FPC2

Configuring IP-AFT Prefix Filtering

In the OpenConfig Abstract Forwarding Table (OC-AFT) model, the Forwarding Information Base Telemetry Daemon (FIBTD) not only translates the FIB state to the external collector, but also enables you to configure prefix filtering on the target device (source). With prefix filtering, you control the flow of data from your device to the collector. Only interfaces with the required prefixes and their corresponding next-hop and next-hop group containers are exported to the OC-AFT collector.

Reducing the set of interfaces to only those with prefixes of interest decreases the overall CPU and resource usage on Routing Engines, FPCs, and MPCs.

To configure prefix filtering:

1. In configuration mode, go to the `[edit system fib-streaming]` hierarchy level and add the forwarding information base (FIB) table you want to target using the `table` statement.

```
[edit]
user@host# set system fib-streaming prefix-list table FIB1
```

2. Include the `family` statement and specify either `inet` (IPv4) or `inet6` (IPv6).

```
[edit system fib-streaming prefix-list table FIB1]
user@host# set family inet6
```

3. Use the `prefix` statement to add the address prefix following; here, the address is `2001:db8:0000:0000:250:af:34ff:fe26/64` and the prefix is `2001:db8:0000:0000:250:af:34ff:fe26/64`

```
[edit system fib-streaming prefix-list table FIB1 family inet6]
user@host# set prefix 2001:db8:0000:0000:250:af:34ff:fe26/64
```

4. In Junos OS, to verify if prefix filtering is configured, use the operational mode command `show fib-streaming state` to display the Operation mode. The value `prefix-filter` confirms that prefix filtering mode is enabled.

See below the sample `show` command output in Junos OS:

```
user@host> show fib-streaming state
Jun 03 22:45:43
    FIB states resync: 1
    RPD client session state: Connected
    LSP name init sync done: 1
```



```
Resync done: 1
Operation mode: prefix-filter
```

In Junos OS Evolved, the value `prefix-filter-on-change` confirms that prefix filtering mode is enabled, else the value is `on-change` for non-prefix filtering mode.



NOTE: For Junos OS: Export data from your source (target) to a collector using the sensor `/network-instances/network-instance/afts/`. When using prefix-filter mode, we recommend you use streaming subscription mode at a sampling frequency of 5 minutes or more and that all collectors have the same sampling frequency. If you are using non-prefix-filter mode, we recommend you use `ON_CHANGE` subscription mode. You can also export data using User Datagram Protocol (UDP).

For Junos OS Evolved: The recommended usage is always `ON_CHANGE` subscription mode, for both prefix-filtering and non-prefix filtering mode.

RELATED DOCUMENTATION

prefix-list (Junos Telemetry Interface)

show fib-streaming state

gRPC Services for Junos Telemetry Interface

Configure a NETCONF Proxy Telemetry Sensor in Junos

Enable Subscriber and Service Accounting for BNG CUPS Telemetry

Junos Telemetry Interface (JTI) allows streaming of accounting statistics information on the BNG CUPS Controller (Broadband Network Gateway Control and User Plane Separation) by subscribing to the subscribers sensor. The statistics information includes actual subscriber transit statistics, firewall filter statistics, and subscriber metadata.

Use the following configuration statements to enable telemetry support for subscriber metadata:

1. Specify the `subscriber-info` configuration statement to enable streaming of subscriber metadata to an external telemetry server when the subscriber logs into or logs out of the device under test (DUT).

```
set dynamic-profiles CLIENT-PROFILE telemetry subscriber-info
```
2. Specify the `subscriber-statistics` option to allow periodic streaming of subscriber accounting statistics such as input packets/octets and output packets/octets to an external telemetry server.

```
set dynamic-profiles CLIENT-PROFILE telemetry subscriber-info subscriber-statistics
```

3. Specify the service-statistics option to allow periodic streaming of firewall filter statistics for IPv4 and IPv6 filters to an external telemetry server.

```
set dynamic-profiles FILTER-PROFILE telemetry subscriber-info service-statistics
```

The subscriber-info hierarchy level contains the configuration commands to enable telemetry for subscriber and service statistics. These commands were previously under the dynamic-profiles hierarchy.

Use the following command to check the sensor subscription:

```
user@host> show agent sensors
Sensor Information Sensor name      :__default_fabric_sensor__
Resource           :/junos/system/linecard/fabric
Object ID          :2277069355
Sensor ID           :9288676508521003
Polling Interval    :2000000000
Server Information Server name      :__default__snmp_server__
Server ID :33
Scope ID :0
Remote-Address :0.0.0.0
Remote-port :0
Profile Information: Profile
name           :__default_snmp_export_profile__
profile ID :1
Rep-interval :2000
Local-Address :0.0.0.0
Local-port :0
Timestamp :0
Format :1
Transport :0
Sensor Information Sensor name :ddos
Resource :/junos/system/linecard/ddos/
Object ID :3440566888
Sensor ID :21110626693866088
Polling Interval :1000
Server Information Server
name :sserver
Server ID :75
Scope ID :0 Remote-
Address :10.0.0.1
Remote-port :2000
Local-Address :10.0.0.10
```

```

Local-port      :21111
Timestamp       :1
Format          :1
Transport       :0

```

The following table describes the sensor subscription path and their data models:

Table 17: Data models and sensor subscription paths

Data Exported	Sensor Path	Usage Group	Data Model
Subscriber information	<i>/junos/system/ subscriber- management/ cups/cp- instances/cp- instance/ subscribers/ subscriber-info</i>	<i>interface- information</i> group	<p>Uses the data model as in Integrated BNG, except data streamed is CP specific.</p> <p>There are additional fields added to the <i>interface-information</i> group:</p> <p>PeerSid – This field gives the session ID on the user plane.</p> <p>User-plane:port – This field gives the user plane name and port on which the session has come up.</p> <p>SGRP Name– This field gives the SGRP (subscriber group) name of the session.</p> <p>Device ID– This field gives the device ID of the session.</p>
Subscriber statistics	<i>/junos/system/ subscriber- management/ cups/cp- instances/cp- instance/ subscribers/ accounting- statistics</i>	<i>actual-transit- statistics</i> group	The data model remains same as subscriber-statistics in integrated BNG, except for the sensor subscription path.

Table 17: Data models and sensor subscription paths (*Continued*)

Data Exported	Sensor Path	Usage Group	Data Model
Service statistics	<i>/junos/system/ subscriber- management/ cps/cp- instances/cp- instance/ subscribers/service- statistics</i>	<i>service-filter- statistics</i> group	The data model remains same as service-statistics in integrated BNG, except for the sensor subscription path.

JTI exports all the above sensor data to an external collector via telemetry for each control plane.

The configuration hierarchy for the `dynamic-profiles` statement is as follows.

```
telemetry {
    subscriber-info {
        subscriber-statistics;
        service-statistics;
    }
    queue-statistics {
        interface $junos-interface-name {
            refresh rate;
            queues queue set;
        }
        interface-set $junos-interface-set-name {
            refresh rate;
            queues queue set;
        }
    }
}
```



NOTE: The `subscriber-info` configuration statement is mandatory under `telemetry`. Within the `subscriber-info` statement, there are two optional configurations: `service-statistics` and `subscriber-statistics`.

RELATED DOCUMENTATION

subscriber-info

<https://apps.juniper.net/ydm-explorer/>

CPU and NPU Sensor Support for MX Series Routers with MPC10E-15C-MRATE Line Cards

Junos OS Release 19.3R1 supports CPU and network processing unit (NPU) sensors on MX Series routers with MPC10E-10C-MRATE and MPC10E-15C-MRATE line cards. JTI enables the export of statistics from these sensors to outside collectors at configurable intervals using gRPC services.

Unlike the Junos kernel implementation for the CPU and NPU sensors in previous Junos releases, this feature uses the OpenConfig AFT model. Because of this, there is a difference in the resource path and key-value (kv) pair output compared to the Junos kernel output.

Use the following resource path to export statistics:

- `/junos/system/linecard/cpu/memory/`
- `/junos/system/linecard/npu/memory/`
- `/junos/system/linecard/npu/utilization/`

To provision the sensor to export data through gRPC services, use the `telemetrySubscribe` RPC to specify telemetry parameters. Streaming telemetry data through gRPC also requires the OpenConfig for Junos OS module.

For more information about gRPC resource paths, see [Guidelines for gRPC Sensors \(Junos Telemetry Interface\)](#).

Delivery of Telemetry Data for AFT-Based Line Cards on MX Series Routers

Starting with Junos OS Release 20.4R1, a new concatenation of prefix and key delivery for telemetry is implemented for Modular Port Concentrator (MPC) line cards supporting Abstract Forwarding Table (AFT) (such as MPC10E or MPC11E line cards). The OpenConfig path is split in a different manner as compared to the syntax on other line cards.

The output that follows illustrates the changes in prefix and key delivery.

```
Example jtimon output for a port on non-AFT LC
key: __prefix__
str_value: /junos/firewall[name='CORERO-MITIGATE-xe-1/0/0.4051-i']/state/
key: counter[name='Corero-Allowed-xe-1/0/0.4051-i']/packets
uint_value: 2935220

<<RESULT>>
path: /junos/firewall[name='CORERO-MITIGATE-xe-1/0/0.4051-i']/state/counter[name='Corero-Allowed-
xe-1/0/0.4051-i']/packets
uint_value: 2935220

Example jtimon output for a port on an AFT LC
key: __prefix__
str_value: /junos/firewall[name='CORERO-MITIGATE-et-2/0/0.114-i']/state/counter[name='Corero-
Allowed-et-2/0/0.114-i']/
key: packets
uint_value: 29191907

<<RESULT>>
path: /junos/firewall[name='CORERO-MITIGATE-et-2/0/0.114-i']/state/counter[name='Corero-Allowed-
et-2/0/0.114-i']/packets
uint_value: 29191907
```

Diameter Application Protocol And Diameter Peer Sensors For Subscribers

JTI supports streaming statistics for subscribers for the diameter application protocols Network Access Server Application (NASREQ), policy and charging rules function (PCRF), and Online Charging System (OCS). There are also new diameter peer sensors that provide response time measurements for messages exchanged between an MX router and the peer for each of the diameter applications. Statistics are exported using JTI and the Juniper AAA Model, which covers telemetry export using gRPC, gNMI, or Juniper proprietary RPC or UDP.

To stream diameter application statistics, include the resource paths:

- For NASREQ statistics, `/junos/system/subscriber-management/aaa/diameter/clients/nasreq`
- For PCRF statistics, `/junos/system/subscriber-management/aaa/diameter/clients/gx`
- For OCS statistics, `/junos/system/subscriber-management/aaa/diameter/clients/gy`

To stream response time measurements for the diameter applications, include the resource paths in a subscription or using the sensor configuration statement:

- For NASREQ measurements, `/junos/system/subscriber-management/aaa/diameter/peers/peer[peer_address=peer-address]/nasreq/response-time`
- For PCRF measurements, `/junos/system/subscriber-management/aaa/diameter/peers/peer[peer_address=peer-address]/gx/response-time`
- For OCS measurements, `/junos/system/subscriber-management/aaa/diameter/peers/peer[peer_address=peer-address]/gy/response-time`

To enable these statistics for an MX Series router for native (UDP) export, include the `sensors` statement at the `[edit services analytics]` hierarchy level.

To provision the sensor to export data through gNMI, use the `Subscribe` RPC defined in the [gnmi.proto](#) to specify request parameters.

To provision the sensor to export data through gRPC, use the `telemetrySubscribe` RPC to specify telemetry parameters. Streaming telemetry data through gRPC also requires the OpenConfig for Junos OS module.

Junos OS Release 19.3R1 supports diameter application protocol sensors for MX5, MX10, MX40, MX150, MX204, MX240, MX480, MX960, MX2008, MX2010, MX2020, MX10003, MX10008, and MX100016 routers.

For more information about gRPC and gNMI resource paths, see [Guidelines for gRPC and gNMI Sensors \(Junos Telemetry Interface\)](#).

Dynamic Tunnel Statistics Support

Starting with Junos OS Release 17.4R1, you can export counter statistics for Packet Forwarding Engine dynamic tunnels to an outside collector using either native (UDP) or OpenConfig telemetry sensors through JTI.

The statistics are used to report various network element performance metrics in a scalable and efficient way, providing visibility into Packet Forwarding Engine errors and drops.

A timestamp indicating when the counters were last reset is included with all the exported data to allow collectors to determine if and when a reset event happened; for example, if the Packet Forwarding Engine hardware restarted.

Exported statistics are similar to the output of the operational mode command `show nhdb hw dynamic-ip-tunnels`.

To provision statistics export through gRPC, use the `telemetrySubscribe` RPC to create a subscription and specify telemetry parameters. Include the resource path `/junos/services/ip-tunnel[name='tunnel-name']/usage/counters[name='counter-name']/` in the subscription.

Streaming telemetry data through gRPC also requires the OpenConfig for Junos OS module.

To configure export of statistics through UDP, include the sensor `/junos/services/ip-tunnel/usage/` in the [sensor \(Junos Telemetry Interface\)](#) configuration statement at the `[edit services analytics]` hierarchy level. All parameters for UDP sensors are configured at that hierarchy level. MX80 and MX104 routers support only UDP streaming. They do not support gRPC.

Enabling Streaming of Telemetry Sensor Information for SR-TE policies (BGP or Static)

Starting with Junos OS Release 18.3R1, OpenConfig support for MX Series and PTX Series through gRPC and JTI provides continuous statistics streaming via the same sensor irrespective of the route that is active (BGP or static) for a given Segment Routing Traffic Engineering (SR-TE) policy.

Support is available in Junos OS Evolved Release 21.4R1EVO for PTX10001-36MR, PTX10004, PTX10008, and PTX10016 routers.

This feature provides support for BGP [DRAFT-SRTE] and statically configured SR-TE policies at ingress routers.

To provision the sensor to export data through gRPC streaming, use the `telemetrySubscribe` RPC to specify telemetry parameters. Include the resource path `/mpls/signaling-protocols/segment-routing/` to export these statistics.

In addition to configuring the sensor, you must enable statistics collection through the Junos OS. To do this, include the statistics configuration statement at the `[edit protocols source-packet-routing telemetry]` hierarchy level. Optionally, you can limit statistics by including the `no-transit` or `no-ingress` parameter.

See *Configure a NETCONF Proxy Telemetry Sensor in Junos* for instructions on configuring a sensor.

See [Guidelines for gRPC and gNMI Sensors \(Junos Telemetry Interface\)](#) for further information about resource paths.

Exporting Packet Forwarding Engine Traffic Sensor Data

Starting with Junos OS Release 17.4R1, you can export Packet Forwarding Engine traffic statistics through the Junos telemetry interface for MX Series and PTX Series routers. Both UDP and gRPC are supported.

This sensor tracks reporting of Packet Forwarding Engine statistics counters and provides visibility into Packet Forwarding Engine error and drop statistics. The resource name for the sensor is `/junos/system/linecard/packet/usage/`. The OpenConfig paths report data specific to CPU, NPU and center chip (CC). The following paths are supported:

- `/components/component[name='FPC.id:NPU.id']/properties/property[name='counter']/state/value`, where FPC refers to the Flexible PIC Concentrator and NPU refers to the network processing unit (packet forwarding engine). A sample resource path is `/components/component[name='FPC0:NPU3']/properties/property[name='ts-output-pps']/state/value` where `hwds-data-error` is the counter for Hardware Discards: Data Error.
- `/components/component[name='FPC.id:CC.id']/properties/property[name='counter']/state/value`, where FPC refers to the Flexible PIC Concentrator and CC refers to the center chip. A sample resource path is `/components/component[name='FPC0:CC1']/properties/property[name='lpbk-packets']/state/value` where `lpbk-packets` is the count of Forward packets specific to FPC0, center chip 1.
- `/components/component[name='FPC.id']/properties/property[name='counter']/state/value`, where FPC refers to the Flexible PIC Concentrator. A sample resource path is `/components/component[name='FPC0']/properties/property[name='lts-input-packets']/state/value` where `lts-input-packets` is the CPU counter Local packets input.

To provision the sensor to export data through gRPC, use the `telemetrySubscribe` RPC to specify telemetry parameters. For streaming through UDP, all parameters are configured at the `[edit services analytics]` hierarchy level.

The following is a map of counters to output fields in the `show pfe statistics traffic` command or `show pfe statistics traffic detail` command (supported only on MX Series routers).

```
CPU stats: (FPCX:CPUY)
```

```
Packet Forwarding Engine local traffic statistics:
```

Local packets input	:	2
Local packets output	:	1
Software input control plane drops	:	0
Software input high drops	:	0
Software input medium drops	:	0

Software input low drops	:	0
Software output drops	:	0
Hardware input drops	:	0

Counter

lts-input-packets	Local packets input
lts-output-packets	Local packets output
lts-sw-input-control-drops	Software input control plane drops
lts-sw-input-high-drops	Software input high drops
lts-sw-input-medium-drops	Software input medium drops
lts-sw-input-low-drops	Software input low drops
lts-sw-output-low-drops	Software output drops

NPU stats: (FPCX:CCY)

Input packets:	1169	0 pps
Output packets:	0	0 pps
Fabric Input :	277235149	16078 pps
Fabric Output :	277235149	16079 pps

Counter

ts-input-packets	Input packets
ts-input-packets-pps	Input packets in pps
ts-output-packets	Output packets
ts-output-packets-pps	Output packets in pps
ts-fabric-input-packets	Fabric Input
ts-fabric-input-packets-pps	Fabric Input in pps
ts-fabric-output-packets	Fabric Output
ts-fabric-output-packets-pps	Fabric Output in pps

Packet Forwarding Engine loopback statistics:

Forward packets :	0	0 pps
Forward bytes :	0	0 bps
Drop packets :	0	0 pps
Drop bytes :	0	0 bps

Counter

lpbk-packets	Forward packets
lpbk-packets-pps	Forward packets pps
lpbk-packets-byte	Forward bytes

lpbk-packets-bps	Forward bytes	bps
lpbk-drop-packets	Drop packets	
lpbk-drop-packets	Drop packets	pps
lpbk-drop-packets	Drop bytes	
lpbk-drop-packets	Drop bytes	bps
Lu chips stats: FPCx:NPUY		
Counter		
lts-hw-input-drops		
hwds-normal	Hardware discards normal	discard
hwds-fabric	Hardware discards fabric	drops
hwds-info-cell	Hardware discards info	cell drops
hwds-timeout	Hardware discards	timeour
hwds-truncated-key	Hardware discards	truncated key
hwds-bits-to-test	Hardware discards	bits to test
hwds-stack-underflow	Hardware discards	stack underflow
hwds-stack-overflow	Hardware discards	stack overflow
hwds-data-error	Hardware discards	data error
hwds-extended	Hardware discards	extended discard
hwds-invalid-iif	Hardware discards	invalid interface
hwds-input-checksum	Hardware discards	input checksum
hwds-output-mtu		
hwds-inet-bad-route		
hwds-inet6-bad-route		
hwds-filter-discard		
hwds-dlu-not-routable		

Export Timing Data to Collectors

IN THIS SECTION

- [Configure Timing Data Export | 196](#)
- [Platform-Specific Export Timing Data to Collectors Behavior | 196](#)

Junos telemetry interface (JTI) supports export of timing attributes for Precision Time Protocol (PTP) and Synchronous Ethernet (SyncE) to the collector. Export of data is through native models (export of PTP data is through the YANG model). Both periodic streaming and on change notifications are supported. The following new subscription paths are introduced:

- For PTP: `/state/protocols/ptp/instances/instance[instance-index]/`
- For SyncE: `/state/protocols/synee/`



NOTE:

1. Only gNMI is supported.
2. For periodic streaming the minimum interval is "2" seconds and maximum interval is "65535" seconds.
3. UDP streaming telemetry is not supported.

Configure Timing Data Export

To configure export of timing data, run the following extension service commands on the device:

1. `set system services extension-service request-response grpc ssl local-certificate certificate-id`
2. `set system services extension-service request-response grpc ssl port 32767`
3. `set system services extension-service notification allow-clients address 0.0.0.0/0`
4. Disable zero suppression, run the command `set services analytics zero-suppression no-zero-suppression`

Platform-Specific Export Timing Data to Collectors Behavior

Use [Feature Explorer](#) to confirm platform and release support for specific features.

Use the following table to review platform-specific behaviors for your platforms:

Table 18: Export Timing Data to Collectors

Platform	Difference
PTX10016	<ul style="list-style-type: none"> • PTX10016 supports only SyncE. PTP is not supported on PTX10016. • The following leaves are not supported on PTX: <ul style="list-style-type: none"> • /ptp/instances/instance[instance-index]/ports/port[index]/unicast-negotiation-port-ds/enable • /ptp/instances/instance[instance-index]/globalinfo-ds/e2e-transparent-clock-ds/clock-state

FPC and Optics Support

Starting in Junos OS Release 19.2R1, JTI supports streaming of Flexible PIC Concentrator (FPC) and optics statistics for the MX Series using Remote Procedure Calls (gRPC). gRPC is a protocol for configuration and retrieval of state information. Support includes the addition of a new process (SensorD daemon) to export telemetry data for integration with AFTTelemetry and LibTelemetry libraries in the OpenConfig model called AFT platform.

The following base resource paths are supported:

- /junos/system/linecard/environment/
- /junos/system/linecard/optics/

To provision the sensor to export data through gRPC, use the telemetrySubscribe RPC to specify telemetry parameters. Streaming telemetry data through gRPC also requires the OpenConfig for Junos OS module.

Interface Burst Monitoring

Junos OS Evolved Release 19.3R1 supports interface burst monitoring on Junos telemetry interface (JTI) to monitor physical interfaces for bursts on QFX5220-128C and QFX5220-32CD switches. Use interface burst monitoring to help troubleshoot problems, make decisions, and adjust resources as needed.

The sampling is done in the millisecond granularity during the export interval (window). The export interval is configured in the sensor with the subscription from the collector. When the sensor is installed, a timer is started in the Packet Forwarding Engine to poll the hardware in 30-100ms intervals. Rates in the first export batch will be 0.

The peak byte is the average of the number of bytes seen in a sampling interval. For bursts lasting less than the sampling interval, the peak byte is averaged out over the interval. Exported statistics also include the time peak bytes are detected, as well as the direction (transmit or receive). The maximum byte rate detected during the export interval among all the samples is considered as the burst. If there are multiple bursts of the same number of bytes rate in the interval, then the first occurring burst is considered as the maximum burst and the timestamp of that burst is considered as the burst timestamp.

Data for all physical interfaces that are UP is exported. Aggregate interfaces are not supported.

You can export interface burst statistics from the Juniper device to an outside collector by including the sensor `/junos/system/linecard/bmon-sw/` in a subscription using remote procedure call (gRPC) services. Only one collector is supported with this sensor.

To provision the sensor to export data through gRPC services, use the `telemetrySubscribe` RPC to specify telemetry parameters. Streaming telemetry data through gRPC also requires the OpenConfig for Junos OS module.



NOTE: This feature does not detect microbursts.

Interface Express Sensor

The interface express sensor is supported by JTI to export interface operational UP and DOWN status at a user-configurable rate. This sensor leverages statistics out of the physical interface sensor, providing faster and more frequent operational status statistics. Only the physical interfaces' operational status from the Flexible PIC Concentrator (FPC) is collected and reported. Statistics from the Routing Engine interface are not reported.

You can use the sensor to export statistics either through UDP (native) export or through gRPC services.

For either export method, include the following resource path:

- `/junos/system/linecard/intf-exp/`

Junos OS Release 18.1R1 supports interface express sensor for PTX1000, PTX3000, PTX5000, and PTX10000 routers.

Junos OS Release 19.3R1 supports interface express sensor for MX960, MX2010, and MX2020 routers.

For more information about gRPC resource paths, see [Guidelines for gRPC and gNMI Sensors \(Junos Telemetry Interface\)](#).

JTI Broadband Edge Statistics Support for Junos Fusion on MX Series

Starting in Junos OS Release 19.2R1, subscriber-based telemetry streaming is enabled when an MX router is configured for Broadband Network Gateway (BNG) and Junos Fusion where subscribers are connected through Junos Fusion Satellite devices. You can use remote procedure calls (gRPC) to export broadband edge (BBE) telemetry statistics to external collectors.

You can stream all BBE resource paths except for the following:

- `/junos/system/subscriber-management/access-network/ancp`
- `/junos/system/subscriber-management/client-protocols/l2tp`
- `/junos/system/subscriber-management/infra/network/l2tp/`

To stream BBE statistics, include a resource path starting with `/junos/system/subscriber-management/` in your gRPC subscription.

To provision the sensor to export data through gRPC, use the `telemetrySubscribe` RPC to specify telemetry parameters.

Physical Ethernet Interface Sensor

Junos OS Release 19.4R1 supports physical Ethernet interface statistics for ON_CHANGE and streaming statistics using JTI and gRPC services or gNMI services on MX960, MX2020, PTX1000, and PTX5000 routers.

This feature supports OpenConfig model **openconfig-if-ethernet.yang** (physical interface level) version 2.6.2 (no configuration).

Use the base resource path **/interfaces/interface/ethernet/state/** in a gRPC or gNMI subscription to export statistics from the Juniper device to an outside collector.

Sensor Power-State Management Support Using gNMI

You can use gNMI to manage the power state of the Routing Engines (controller cards) on a dual Routing Engine system. You can configure a Routing Engine to power off and remain powered off by using gNMI to set the OpenConfig path **/components/component/controller-card/config/power-admin-state** value to **POWER_DISABLED**.

When you configure the power state of the backup Routing Engine (secondary controller card) as **POWER_DISABLED**, the configuration takes effect immediately. The device powers off the Routing Engine and sets the **config/power-admin-state** to **POWER_DISABLED**. The Routing Engine remains powered off even through reboots. Similarly, setting the power state to **POWER_ENABLED** immediately brings the Routing Engine back online.

If you configure the power state of the primary Routing Engine as **POWER_DISABLED**, the configuration takes effect upon the next reboot or switchover. However, configuring **POWER_DISABLED** on the primary Routing Engine automatically triggers a switchover. Thus, after the primary Routing Engine assumes the backup role, it immediately powers off.

Consider the following rules before making changes to the power-admin-state of your device:

- Only controller cards in the **SECONDARY** state/redundant-role honor changes in the **config/power-admin-state** to **POWER_DISABLED**.
- On controller cards in the **PRIMARY** state/redundant-role, if you try to set **config/power-admin-state** to **POWER_DISABLED**, the operation is allowed. However, the **state/power-admin-state** may remain as **POWER_ENABLED** until the next switchover or reboot.

- A switchover is initiated to change the role of **PRIMARY** to **SECONDARY** as soon as you configure the active Routing Engine as **POWER_DISABLED**.

A gNMI client can configure a Routing Engine to remain powered off, for example:

```
gnmic -a hostname:port -u user -p password --tls-ca /path/to/serverRootCA --update-path /
components/component[name=node]/controller-card/config/power-admin-state/ --update-value
"POWER_DISABLED" --timeout timeout
```

The following example uses the gNMI client `gnmic` to send a SetRequest to update the `config/power-admin-state` value to `POWER_DISABLED` for the backup Routing Engine (RE1):

```
user@client:~$ gnmic -a 10.1.1.2:32767 -u grpc-user -p password --tls-ca /etc/pki/certs/
serverRootCA.crt set --update-path /components/component[name=RE1]/controller-card/config/power-
admin-state/ --update-value "POWER_DISABLED" --timeout 30s
{
  "source": "10.1.1.2:32767",
  "timestamp": 1739315808281370579,
  "time": "2025-02-11T15:16:48.281370579-08:00",
  "results": [
    {
      "operation": "UPDATE",
      "path": "openconfig:components/component[name=RE1]/controller-card/config/power-admin-
state"
    }
  ]
}
```

Verify the configuration.

```
user@host> show configuration openconfig-platform:components
component RE1 {
  controller-card {
    config {
      openconfig-platform-controller-card:power-admin-state POWER_DISABLED;
    }
  }
}
```

The OpenConfig configuration is equivalent to issuing the `set system node offline node` configuration mode command in the Junos OS Evolved configuration.

Use the `show system nodes operational mode` command to verify that the state of the Routing Engine is offline.

```
user@host> show system nodes re1
Node: re1
  Node Id    : 0
  Node Nonce : 0
  Status     : offline, configured-offline
  Attributes :
```

Similarly, you can re-enable the Routing Engine.

```
gnmic -a hostname:port -u user -p password --tls-ca /path/to/serverRootCA --update-path /
components/component[name=node]/controller-card/config/power-admin-state/ --update-value
"POWER_ENABLED" --timeout timeout
```

The following example uses the gNMI client `gnmic` to send a `SetRequest` to update the `config/power-admin-state` value to `POWER_ENABLED` for the backup Routing Engine (RE1) :

```
user@client:~$ gnmic -a 10.1.1.2:32767 -u grpc-user -p password --tls-ca /etc/pki/certs/
serverRootCA.crt set --update-path /components/component[name=RE1]/controller-card/config/power-
admin-state/ --update-value "POWER_ENABLED" --timeout 30s
{
  "source": "10.1.1.2:32767",
  "timestamp": 1739315506688244303,
  "time": "2025-02-11T15:11:46.688244303-08:00",
  "results": [
    {
      "operation": "UPDATE",
      "path": "openconfig:components/component[name=RE1]/controller-card/config/power-admin-
state"
    }
  ]
}
```

When you update the `config/power-admin-state` to `POWER_ENABLED`, the Routing Engine is brought online.

```
user@host> show system nodes re1
Node: re1
  Node Id    : 2201170739205
```

```

Node Nonce : 2159053780
Status      : online, apps-ready
Attributes  : FABRIC_CONTROL (Spare), FABRIC_FCHIP_PARALLEL (Spare), PFE_TOKEN (Spare), RE
              (Spare), TIMINGD_RE (Spare), BackupRE (Active)

```

Standby Routing Engine Sensors For Subscribers

JTI supports streaming standby Routing Engine statistics using gRPC services. This feature is supported on both single chassis and virtual chassis unless otherwise indicated. Use this feature to better track the state of software components running on a standby Routing Engine. Statistics exported to an outside collector through the following sensors (primarily under subscriber management) provide a more complete view of the system health and resiliency state:

- Chassis role (backup or primary) sensor `/junos/system/subscriber-management/chassis` and `/junos/system/subscriber-management/chassis[chassis-index=chassis-index]` (for specifying an index for an MX Series Virtual Chassis)
- Routing Engine status and GRES notification sensor `/junos/system/subscriber-management/chassis/routing-engines/routing-engine` and `/junos/system/subscriber-management/chassis/routing-engines/routing-engine[re-index=RoutingEngineIndex]` (to specify an index number for a specific Routing Engine)
- Subscriber management process sensor `/junos/system/subscriber-management/chassis/routing-engines/process-status/subscriber-management-processes/subscriber-management-process` and `/junos/system/subscriber-management/chassis/routing-engines/process-status/subscriber-management-processes/subscriber-management-process[pid=ProcessIdentifier]` (to specify a PID for a specific process)
- Per Routing Engine DHCP binding statistics for server or relay sensor `/junos/system/subscriber-management/chassis/routing-engines/routing-engine/dhcp-bindings/dhcp-element[dhcp-type-name=RelayOrServer/v4]` and `/junos/system/subscriber-management/chassis/routing-engines/routing-engine/dhcp-bindings/dhcp-element[dhcp-type-name=RelayOrServer/v6]`
- Virtual Chassis port counter sensor `/junos/system/subscriber-management/chassis/virtual-chassis-ports/virtual-chassis-port` and `/junos/system/subscriber-management/chassis/virtual-chassis-ports/virtual-chassis-port[vcp-interface-name=vcp-interface-port-string]` (to specify the interface name). This resource path is only supported on a virtual chassis.

Junos OS Release 20.2R1 supports standby Routing Engine sensors for MX480, MX960, MX10003, MX2010, and MX2020 routers.

For more information about gRPC and gNMI resource paths, see [Guidelines for gRPC and gNMI Sensors \(Junos Telemetry Interface\)](#).

Support For LSP Statistics

You can provision the LSP statistics sensor `/junos/services/label-switched-path/usage/` to monitor per-MPLS LSP statistics. Telemetry data is streamed from Junos devices and exported through JTI to external collectors at configurable intervals through gRPC without involving polling.

Initial support of this feature in Junos OS Release 15.1F6 supported ingress LSPs only when a subscription was made to `/junos/services/label-switched-path/usage/`. With bypass support added to this feature in Junos OS Release 17.4R1, this subscription now streams both ingress LSP and bypass LSP statistics to a collector.

Statistics that are streamed are similar to the output displayed by the operational mode commands `show mpls lsp bypass statistics` and `show mpls lsp ingress statistics`.

For bypass LSPs, the following are exported:

- Bypass LSP originating at the ingress router of the protected LSP.
- Bypass LSP originating at the transit router of the protected LSP.
- Bypass LSP protecting the transit LSP as well as the locally originated LSP.

When the bypass LSP is active, traffic is exported both on the bypass LSP and the ingress (protected) LSP.

To provision a sensor to export data through gRPC, use the `telemetrySubscribe` RPC to specify telemetry parameters. Streaming telemetry data through gRPC also requires the OpenConfig for Junos OS module.

See [Configuring a Junos Telemetry Interface Sensor \(CLI Procedure\)](#) for information about configuring a UDP (native) sensor.

See [Table 19 on page 205](#) for the level of LSP sensor support by platform.

Table 19: LSP Support by Platform

Platform	Ingress LSP, UDP Feature Introduced	Ingress LSP, gRPC Streaming Feature Introduced	Bypass LSP Feature Introduced
ACX6360			Junos 19.2R1 RSVP bypass LSP origin transit node
MX80/MX104	Junos OS Release 15.1F6 Junos OS Release 16.1R3 Junos OS Release 17.2R1		Junos OS Release 17.4R1 Junos OS Release 17.2R1
MX Series with MPC	Junos OS Release 15.1F6	Junos OS Release 16.1R4 Junos OS Release 17.2R1	Junos OS Release 17.4R1 Junos OS Release 17.2R1
PTX5000 with FPC3		Junos OS Release 18.2R1	Junos OS Release 17.4R1
PTX3000 with FPC3	Junos OS Release 15.1F6 Junos OS Release 16.1R3 Junos OS Release 17.2R1	Junos OS Release 16.1R4 Junos OS Release 17.2R1 Junos OS Release 18.2R1	Junos OS Release 17.4R1 Junos OS Release 17.2R1
PTX Series with FPC1/2	Junos OS Release 15.1F6 Junos OS Release 16.1R3 Junos OS Release 17.2R1	Junos OS Release 16.1R4 Junos OS Release 17.2R1 Junos OS Release 18.2R1	Junos OS Release 17.4R1 Junos OS Release 17.2R1
PTX1000	Junos OS Release 16.1R3	Junos OS Release 16.1R4 Junos OS Release 17.2R1	Junos OS Release 17.4R1 Junos OS Release 17.2R1
PTX10000	Junos OS Release 17.3R1	Junos OS Release 17.3R1	Junos OS Release 17.4R1 Junos OS Release 17.2R1

Table 19: LSP Support by Platform *(Continued)*

Platform	Ingress LSP, UDP Feature Introduced	Ingress LSP, gRPC Streaming Feature Introduced	Bypass LSP Feature Introduced
PTX10001-20C			Junos OS Release 19.1R1 RSVP bypass LSP origin transit node
PTX10002	Junos OS Release 19.1R1	Junos OS Release 19.1R1	
VMX	Junos OS Release 17.3R1	Junos OS Release 17.3R1	Junos OS Release 17.4R1 Junos OS Release 17.2R1
MX150	Junos OS Release 17.4R1	Junos OS Release 17.4R1	
EX4600	Junos OS Release 18.4R1		
EX4650	Junos OS Release 18.3R1	Junos OS Release 18.3R1	
EX9200	Junos OS Release 17.3R1		
QFX10000			
QFX5200	Junos OS Release 17.2R1	Junos OS Release 17.2R1	
QFX10002	Junos OS Release 19.1R1	Junos OS Release 19.1R1	
QFX5100	Junos OS Release 18.2R1	Junos OS Release 18.2R1	
QFX5110	Junos OS Release 18.2R1	Junos OS Release 18.2R1	
QFX5120-48Y	Junos OS Release 18.3R1	Junos OS Release 18.3R1	

Table 19: LSP Support by Platform *(Continued)*

Platform	Ingress LSP, UDP Feature Introduced	Ingress LSP, gRPC Streaming Feature Introduced	Bypass LSP Feature Introduced
QFX5200	Junos OS Release 18.2R1	Junos OS Release 18.2R1	

NPU Sensors

IN THIS SECTION

- [NPU and Firewall Resource Utilization | 207](#)

NPU and Firewall Resource Utilization

IN THIS SECTION

- [NPU Utilization for PTX10000 and QFX1000 Series | 208](#)
- [NPU Utilization for PTX Series | 212](#)
- [Firewall Resource Utilization | 221](#)

NPU and resource utilization sensors provide visibility into the internal operations of Juniper devices and the state of resource consumption on each device. You can use this information to improve network design and to optimize traffic engineering. This data can also be used for early detection of problems in individual devices as well as in the overall network and the traffic carried by it.

 **NOTE:** The Juniper Networks

Junos YANG Data Model Explorer

provides a complete list of sensors and resource paths supported by each platform. It enables you to explore or compare various OpenConfig and Native data model attributes. Use the filter option based on the software release number or product to view the list of resource paths and sensors on each platform.

NPU Utilization for PTX10000 and QFX1000 Series

Table 20: NPU Utilization Sensor (resource path /junos/system/linecard/npu/utilization/)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range	Description
NPU Utilization	NPU Utilization	32-bit	1-100	Number on a scale of 0-100 that indicates the busyness of an NPU.
Memory load <ul style="list-style-type: none"> Name 	Memory load <ul style="list-style-type: none"> Name 	—	—	Load on a memory subsystem of the NPU
Memory load	Memory load	—	—	Load on a memory subsystem of the NPU
Memory load <ul style="list-style-type: none"> Name 	Memory load <ul style="list-style-type: none"> Name 	string	—	A name string to identify the particular memory subsystems (such as hmc)
Memory load <ul style="list-style-type: none"> Average_util Highest_util Lowest_util 	Memory load <ul style="list-style-type: none"> Average_util Highest_util Lowest_util 	32-bit	—	Various memory utilization metrics

Table 20: NPU Utilization Sensor (resource path /junos/system/linecard/npu/utilization/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range	Description
Memory load <ul style="list-style-type: none"> Average_cache_hit_rate Highest_cache_hit_rate Lowest_cache_hit_rate 	Memory load <ul style="list-style-type: none"> Average_cache_hit_rate Highest_cache_hit_rate Lowest_cache_hit_rate 	—	—	Each memory is front ended by a cache. The following metrics indicate how these caches are working
Packet Load	Packet Load	—	—	Offered packet load on an internal subsystem of the NPU, like the following: <ul style="list-style-type: none"> loopback_pps recirculation_pps wan_and_host_inject_pps asic_to_host_pps
Packet Load: <ul style="list-style-type: none"> Identifier 	Packet Load: <ul style="list-style-type: none"> Identifier 	string	—	Each internal subsystem of the NPU has a name
Packet Load: <ul style="list-style-type: none"> rate 	Packet Load: <ul style="list-style-type: none"> rate 	64-bit	—	Rate of packets received

Table 20: NPU Utilization Sensor (resource path /junos/system/linecard/npu/utilization/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range	Description
Packet Load: <ul style="list-style-type: none"> • average_instructions_per_packet • average_wait_cycles_per_packet • average_cycles_per_packet 	Packet Load: <ul style="list-style-type: none"> • average_instructions_per_packet • average_wait_cycles_per_packet • average_cycles_per_packet 	32-bit	—	Indicate the compute load on the NPU. These metrics are not valid for the PF chip on the PTX10000 routers or QFX10000 switches.

Table 21: NPU Memory Sensor for PTX10000 and QFX1000 Series (resource path /junos/system/linecard/npu/memory/)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range	Description
Memory Summary	Memory Summary	-	-	NPU memory utilization summary per memory type
Memory Summary <ul style="list-style-type: none"> • Resource_name 	Memory Summary <ul style="list-style-type: none"> • Resource_name 	-	string	A name string to identify the particular memory blocks such as KHT (cuckoo hash tables), edf, flt, sfm, fcv. Beta-0, beta-1, policer, and pclt
Memory Summary: <ul style="list-style-type: none"> • size 	Memory Summary: <ul style="list-style-type: none"> • size 	64-bit	%	Size memory utilization metrics
Memory Summary: <ul style="list-style-type: none"> • allocated 	Memory Summary: <ul style="list-style-type: none"> • allocated 	64-bit	%	Allocated memory utilization metrics

Table 21: NPU Memory Sensor for PTX10000 and QFX1000 Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range	Description
Memory Summary: • utilization	Memory Summary: • utilization	32-bit	%	Memory utilization metrics
Application memory partition summary		-	-	Detailed statistics for NPU memory partitions per application; examples include plctfilter, plct-ingr-nh, plctegr-nh, plct-rt, and plctmisc.
Application memory partition summary: • Application_name	Application memory partition summary: • Application_name	string	-	Name of the application for which NPU memory is allocated.
Application memory partition summary: • Bytes_allocated • Allocation_count • Free_count	Application memory partition summary: • Bytes_allocated • Allocation_count • Free_count	32-bit	-	Various memory values for allocation and free count.

NPU Utilization for PTX Series

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
NPU Memory <ul style="list-style-type: none"> L2 domain 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-l2domain-allocated mem-util-kht-l2domain-size mem-util-kht-l2domain-utilization 	—	3276 your
NPU Memory <ul style="list-style-type: none"> SLU MY-MAC 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-slu-my-mac-size mem-util-kht-slu-my-mac-allocated mem-util-kht-slu-my-mac-utilization 	entry	3072
NPU Memory <ul style="list-style-type: none"> Forwarding table: edb0 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-dlu-edb0-allocated mem-util-kht-dlu-edb0-size mem-util-kht-dlu-edb0-utilization 	entry	1677
NPU Memory <ul style="list-style-type: none"> Forwarding table: edb1 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-dlu-edb1-allocated mem-util-kht-dlu-edb1-size mem-util-kht-dlu-edb1-utilization 	entry	4194
Firewall / Filter <ul style="list-style-type: none"> Filter instances 	Exported property names: not available	entry	8192

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
Firewall / Filter <ul style="list-style-type: none"> Filter terms 	Exported property names: <ul style="list-style-type: none"> mem-util-flt-action-entries-utilization mem-util-flt-action-entries-allocated mem-util-flt-action-entries-size 	—	6553
Firewall / Filter <ul style="list-style-type: none"> Filter alpha block [0] 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-flt0-size mem-util-kht-flt0-allocated mem-util-kht-flt0-utilization 	—	1310
Firewall / Filter <ul style="list-style-type: none"> Filter alpha block [1] 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-flt1-size mem-util-kht-flt1-allocated mem-util-kht-flt1-utilization mem-util-flt-alpha-1-kht-size mem-util-flt-alpha-1-kht-allocated mem-util-flt-alpha-1-kht-utilization mem-util-flt-alpha-1-bft-0-size mem-util-flt-alpha-1-bft-0-allocated mem-util-flt-alpha-1-plt-size mem-util-flt-alpha-1-plt-allocated mem-util-flt-alpha-1-plt-utilization 	—	1310

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
Firewall / Filter <ul style="list-style-type: none">Filter beta block [0]	Exported property names: <ul style="list-style-type: none">mem-util-beta-0-bank-0-sizemem-util-beta-0-bank-0-allocatedmem-util-beta-0-bank-0-utilizationmem-util-beta-0-bank-1-sizemem-util-beta-0-bank-1-allocatedmem-util-beta-0-bank-1-utilizationmem-util-beta-0-bank-2-sizemem-util-beta-0-bank-2-allocatedmem-util-beta-0-bank-2-utilizationmem-util-beta-0-bank-3-sizemem-util-beta-0-bank-3-allocatedmem-util-beta-0-bank-3-utilizationmem-util-beta-0-bank-4-sizemem-util-beta-0-bank-4-allocatedmem-util-beta-0-bank-4-utilization	—	6553

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
Firewall / Filter <ul style="list-style-type: none"> Filter beta block [1] 	Exported property names: <ul style="list-style-type: none"> mem-util-beta-1-bank-0-size mem-util-beta-1-bank-0-allocated mem-util-beta-1-bank-0-utilization mem-util-beta-1-bank-1-size mem-util-beta-1-bank-1-allocated mem-util-beta-1-bank-1-utilization mem-util-beta-1-bank-2-size mem-util-beta-1-bank-2-allocated mem-util-beta-1-bank-2-utilization mem-util-beta-1-bank-3-size mem-util-beta-1-bank-3-allocated mem-util-beta-1-bank-3-utilization mem-util-beta-1-bank-4-size mem-util-beta-1-bank-4-allocated mem-util-beta-1-bank-4-utilization mem-util-flt-beta-1-bank-5-size mem-util-flt-beta-1-bank-5-allocated mem-util-flt-beta-1-bank-5-utilization 	—	6553

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
Firewall / Filter <ul style="list-style-type: none"> Secondary Facet Match 	Exported property names: <ul style="list-style-type: none"> mem-util-sfm-entries-size mem-util-sfm-entries-allocated mem-util-sfm-entries-utilization 	—	8192
Firewall / Filter <ul style="list-style-type: none"> Special Cover Vector 	Exported property names: <ul style="list-style-type: none"> mem-util-flt-scv-size mem-util-flt-scv-allocated mem-util-flt-scv-utilization 	—	256 n pages
Firewall / Filter <ul style="list-style-type: none"> FCV block [1] 	Exported property names: <ul style="list-style-type: none"> mem-util-fcv-blk-1-size mem-util-fcv-blk-1-allocated mem-util-fcv-blk-1-utilization 	B	6553
Firewall / Filter <ul style="list-style-type: none"> FCV block [2] 	Exported property names: <ul style="list-style-type: none"> mem-util-fcv-blk-2-size mem-util-fcv-blk-21allocated mem-util-fcv-blk-2-utilization 	B	6553

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
Firewall / Filter <ul style="list-style-type: none"> FCV block [3] 	Exported property names: <ul style="list-style-type: none"> mem-util-fcv-blk-3-size mem-util-fcv-blk-3-allocated mem-util-fcv-blk-3-utilization 	B	6553
NPU Memory <ul style="list-style-type: none"> Forwarding table: edb0 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-dlu-edb0-allocated mem-util-kht-dlu-edb0-size mem-util-kht-dlu-edb0-utilization 	entry	1677
NPU Memory <ul style="list-style-type: none"> Forwarding table: edb1 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-dlu-edb1-allocated mem-util-kht-dlu-edb1-size mem-util-kht-dlu-edb1-utilization 	entry	4194
NPU Memory <ul style="list-style-type: none"> Forwarding table: edb0 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-dlu-edb0-allocated mem-util-kht-dlu-edb0-size mem-util-kht-dlu-edb0-utilization 	entry	1677
NPU Memory <ul style="list-style-type: none"> Forwarding table: edb1 	Exported property names: <ul style="list-style-type: none"> mem-util-kht-dlu-edb1-allocated mem-util-kht-dlu-edb1-size mem-util-kht-dlu-edb1-utilization 	entry	4194

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
Firewall / Filter <ul style="list-style-type: none">• Policer IDs	Exported property names: <ul style="list-style-type: none">• mem-util-policer-id-size• mem-util-policer-id-allocated• mem-util-policer-id-utilization	B	1638

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
Firewall / Filter <ul style="list-style-type: none"> • Policer/Counter space 	Exported property names: <ul style="list-style-type: none"> • mem-util-plct-size • mem-util-plct-allocated • mem-util-plct-utilization • mem-util-plct-filter-bytes-allocated • mem-util-plct-filter-allocation-count • mem-util-plct-filter-free-count • mem-util-plct-ing-nh-bytes-allocated • mem-util-plct-ing-nh-allocation-count • mem-util-plct-ing-nh-free-count • mem-util-plct-egr-nh-bytes-allocated • mem-util-plct-egr-nh-allocation-count • mem-util-plct-egr-nh-free-count • mem-util-plct-misc-bytes-allocated • mem-util-plct-misc-allocation-count • mem-util-plct-misc-free-count • mem-util-plct-memory-size • mem-util-plct-memory-allocated • mem-util-plct-memory-utilization 	B	1310

Table 22: NPU Memory Sensor for PTX Series (resource path /junos/system/linecard/npu/memory/) (Continued)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range
Next-hops and encapsulation <ul style="list-style-type: none"> IRP Memory: Load-balancing partition 	Exported property names: <ul style="list-style-type: none"> mem-util-jnh-loadbal-allocated mem-util-jnh-loadbal-size mem-util-jnh-loadbal-utilization 	KWords	128
Next-hops and encapsulation <ul style="list-style-type: none"> IRP Memory: Next-hop partition 	Exported property names: <ul style="list-style-type: none"> mem-util-jnh-loadbal-allocated mem-util-jnh-loadbal-size mem-util-jnh-loadbal-utilization mem-util-jnh-loadbal-utilization 	KWords	256
Next-hops and encapsulation <ul style="list-style-type: none"> IRP Memory: EDF partition 	Exported property names: <ul style="list-style-type: none"> mem-util-edf-public-words-allocated mem-util-edf-public-words-size mem-util-edf-public-words-utilization 	KWords	320
Next-hops and encapsulation <ul style="list-style-type: none"> IRP Memory: MPLS label memory 	Exported property names: <ul style="list-style-type: none"> mem-util-jnh-mpls-allocated mem-util-jnh-mpls-size mem-util-jnh-mpls-utilization 	KWords	128

Table 23: NPU Utilization Sensor for PTX Series (resource path `/junos/system/linecard/npu/utilization/`)

Native Sensor Property Name	gRPC Sensor Property Name	Unit Type	Range	Description
NPU Utilization <ul style="list-style-type: none"> Utilization 	Exported property names: <ul style="list-style-type: none"> util-metric 	percent	—	Current traffic
NPU Utilization <ul style="list-style-type: none"> Packet Load 	Exported property names: <ul style="list-style-type: none"> util-loopback-pps-rate util-recirculation-pps-rate util-asic-to-host-pps-rate util-wan-and-host-inject-pps-rate 	pps	—	Traffic loopback inject, Contr
NPU Utilization <ul style="list-style-type: none"> Memory Load 	Exported property names: <ul style="list-style-type: none"> util-hmc-average-util util-hmc-highest-util util-hmc-lowest-util util-hmc-average-cache-hit-rate util-hmc-highest-cache-hit-rate util-hmc-lowest-cache-hit-rate 	percent/per-sec	—	HMC cache

Firewall Resource Utilization

You can export statistics on firewall resource utilization by subscribing to the OpenConfig resource path `/components/component/integrated-circuit/pipeline-counters/`. The operational state sensors display the total supported filter entries and used entries in terms of count and bytes.

State sensors for firewall resource utilization are:

- `/components/component/integrated-circuit/pipeline-counters/packet/lookup-block/state/acl-memory-used-entries`
- `/components/component/integrated-circuit/pipeline-counters/packet/lookup-block/state/acl-memory-total-entries`

- /components/component/integrated-circuit/pipeline-counters/packet/lookup-block/state/acl-memory-used-bytes
- /components/component/integrated-circuit/pipeline-counters/packet/lookup-block/state/acl-memory-total-bytes

The values of the sensors use the following calculations:

- $\text{acl-memory-used-entries} = \text{round} (\text{UtilizationOfTables [a number from 0 to 100]} / 100 * 64000)$

The value of UtilizationOfTables is the maximum value of the following NPU memory statistics, which are leaves under the native sensor /junos/system/linecard/npu/memory/:

- mem-util-flt-vfilter-utilization
- mem-util-flt-phyfilter-utilization
- mem-util-flt-action-entries-utilization
- mem-util-fcv-blk-1-utilization
- mem-util-fcv-blk-2-utilization
- mem-util-fcv-blk-3-utilization
- mem-util-flt-scv-utilization
- mem-util-beta-0-bank-0-utilization
- mem-util-beta-0-bank-1-utilization
- mem-util-beta-0-bank-2-utilization
- mem-util-beta-0-bank-3-utilization
- mem-util-beta-0-bank-4-utilization
- mem-util-beta-1-bank-1-utilization
- mem-util-beta-1-bank-2-utilization
- mem-util-beta-1-bank-3-utilization
- mem-util-beta-1-bank-4-utilization
- mem-util-beta-1-bank-5-utilization
- mem-util-flt-alpha-1-kht-utilization
- mem-util-flt-alpha-1-plt-utilization
- mem-util-policer-id-utilization

- mem-util-plct-utilization
- acl-memory-total-entries = 64K
- acl-memory-used-bytes = acl-memory-used-entries * 8
- acl-memory-total-bytes = acl-memory-total-entries * 8 = 64k * 8

Telemetry Streaming Support for ZR/ZR+ Optics on MPC 10 Linecards

Starting in Junos OS Release 24.4R1, telemetry data streaming for ZR and ZR+ optics is implemented on the MPC 10 linecards for the following leaves:

```
/components/component/transceiver/config/fec-mode - 'config'

/components/component/transceiver/state/fec-mode - 'leaf'
/components/component/transceiver/state/fec-status - leaf
/components/component/transceiver/state/module-functional-type - 'leaf'
/components/component/transceiver/state/fault-condition - leaf
/components/component/transceiver/state/fec-uncorrectable-blocks - leaf
/components/component/transceiver/state/fec-corrected-bits - leaf
/components/component/transceiver/physical-channels/channel/state/output-frequency - 'leaf'
/components/component/transceiver/physical-channels/channel/state/associated-optical-channel -
'leaf'
/components/component/optical-channel/state/frequency - 'leaf'
/components/component/optical-channel/state/line-port - 'leaf'

/components/component/optical-channel/state/output-power/ - instant/avg/min/max/interval/min-
time/max-time - 'container'
/components/component/optical-channel/state/input-power/ - instant/avg/min/max/interval/min-time/
max-time - 'container'
/components/component/optical-channel/state/laser-bias-current/ - instant/avg/min/max/interval/
min-time/max-time - 'container'
/components/component/optical-channel/state/chromatic-dispersion/ - instant/avg/min/max/interval/
min-time/max-time - 'container'
/components/component/optical-channel/state/second-order-polarization-mode-dispersion/ -
instant/avg/min/max/interval/min-time/max-time - 'container'
/components/component/optical-channel/state/polarization-dependent-loss/ - instant/avg/min/max/
```

```

interval/min-time/max-time - 'container'
/components/component/optical-channel/state/osnr/ - instant/avg/min/max/interval/min-time/max-
time - 'container'
/components/component/optical-channel/state/esnr/ - instant/avg/min/max/interval/min-time/max-
time - 'container'
/components/component/optical-channel/state/carrier-frequency-offset/ - instant/avg/min/max/
interval/min-time/max-time - 'container'
/components/component/optical-channel/state/pre-fec-ber/ - instant/avg/min/max/interval/min-time/
max-time - 'container'
/components/component/optical-channel/state/modulator-bias-xi/ - instant/avg/min/max/interval/
min-time/max-time - 'container'
/components/component/optical-channel/state/modulator-bias-xq/ - instant/avg/min/max/interval/
min-time/max-time - 'container'
/components/component/optical-channel/state/modulator-bias-yi/ - instant/avg/min/max/interval/
min-time/max-time - 'container'
/components/component/optical-channel/state/modulator-bias-yq/ - instant/avg/min/max/interval/
min-time/max-time - 'container'
/components/component/optical-channel/state/modulator-bias-x-phase/ - instant/avg/min/max/
interval/min-time/max-time - 'container'
/components/component/optical-channel/state/modulator-bias-y-phase/ - instant/avg/min/max/
interval/min-time/max-time - 'container'
/components/component/optical-channel/state/sop-roc/ - instant/avg/min/max/interval/min-time/max-
time - 'container'

```

Leaf-Level Support for PFE Sensors

SUMMARY

You can configure leaf-level resource paths for telemetry data export on network devices to reduce computational overhead and bandwidth usage. This feature allows selective querying of specific data elements, such as interface operational status or packet counters, instead of exporting data for all instances of a resource. By focusing on relevant leaves, you obtain precise information efficiently, enhancing performance for telemetry collectors.

Using this leaf level support feature, any path till a given element can be configured and just data corresponding to that element is exported.

The following XPATH leaf options are supported:

- Precise leaf such as `/interfaces/interface[name='et-1/0/35']/state/counters/in-pkts`
- A container within the path such as `/interfaces/interface[name='et-1/0/35']/state/counters/`
- A list within the path `/interfaces/interface/`
- Exported leafs for IFD/IFL/Queues

Following is the full list of leafs that are exported for physical, logical interfaces and their associated queues.

- **IFD or Physical Interfaces**

```
interfaces/interface/state/counters/in-pkts
interfaces/interface/state/counters/in-octets
interfaces/interface/state/counters/in-unicast-pkts
interfaces/interface/state/counters/in-broadcast-pkts
interfaces/interface/state/counters/in-multicast-pkts
interfaces/interface/state/counters/in-pause-pkts
interfaces/interface/state/counters/in-discards
interfaces/interface/state/counters/in-errors
interfaces/interface/state/counters/in-unknown-proto-pkts
interfaces/interface/state/counters/out-pkts
interfaces/interface/state/counters/out-octets
interfaces/interface/state/counters/out-unicast-pkts
interfaces/interface/state/counters/out-broadcast-pkts
interfaces/interface/state/counters/out-multicast-pkts
interfaces/interface/state/counters/out-pause-pkts
interfaces/interface/state/counters/out-discards
interfaces/interface/state/counters/out-errors
interfaces/interface/state/counters/out-unknown-proto-pkts
```

- **IFD Queues**

```
interfaces/interface/state/counters/out-queue/queue-number
interfaces/interface/state/counters/out-queue/pkts
interfaces/interface/state/counters/out-queue/bytes
interfaces/interface/state/counters/out-queue/tail-drop-pkts
interfaces/interface/state/counters/out-queue/tail-drop-bytes
interfaces/interface/state/counters/out-queue/r1-drop-pkts
interfaces/interface/state/counters/out-queue/r1-drop-bytes
```

```

interfaces/interface/state/counters/out-queue/red-drop-pkts
interfaces/interface/state/counters/out-queue/red-drop-bytes
interfaces/interface/state/counters/out-queue/avg-buffer-occupancy
interfaces/interface/state/counters/out-queue/cur-buffer-occupancy
interfaces/interface/state/counters/out-queue/peak-buffer-occupancy
interfaces/interface/state/counters/out-queue/allocated-buffer-size

```

- **IFL or Logical Interfaces**

```

interfaces/interface/subinterfaces/subinterface/state/counters/in-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-octets
interfaces/interface/subinterfaces/subinterface/state/counters/out-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/out-octets
interfaces/interface/subinterfaces/subinterface/state/counters/in-unicast-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-multicast-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/out-unicast-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/out-multicast-pkts

```

- **IFL Queues**

```

interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/queue-number
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/bytes
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/tail-drop-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/tail-drop-bytes
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/r1-drop-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/r1-drop-bytes
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/red-drop-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/red-drop-bytes
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/avg-buffer-occupancy
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/cur-buffer-occupancy
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/peak-buffer-occupancy
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/allocated-buffer-size
interfaces/interface/subinterfaces/subinterface/state/counters/out-queue/queue-number
interfaces/interface/subinterfaces/subinterface/state/counters/out-queue/pkts
interfaces/interface/subinterfaces/subinterface/state/counters/out-queue/bytes
interfaces/interface/subinterfaces/subinterface/state/counters/out-queue/tail-drop-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/r1-drop-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/r1-drop-bytes
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/red-drop-pkts
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/red-drop-bytes

```

```

interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/avg-buffer-occupancy
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/cur-buffer-occupancy
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/peak-buffer-occupancy
interfaces/interface/subinterfaces/subinterface/state/counters/in-queue/allocated-buffer-size
interfaces/interface/subinterfaces/subinterface/state/counters/out-queue/queue-number
interfaces/interface/subinterfaces/subinterface/state/counters/out-queue/pkts
interfaces/interface/subinterfaces/subinterface/state/counters/out-queue/bytes
interfaces/interface/subinterfaces/subinterface/state/counters/out-queue/tail-drop-pkts

```

**NOTE:**

- Leaf level support is applicable to both gRPC and gNMI transport modes.
- Leaf level subscription is not supported for leaves where configuration and exported paths are different.

Resource Filtering

SUMMARY

Resource filtering feature selectively exports telemetry data from specific network device components. This reduces bandwidth usage and processing time.

IN THIS SECTION

- [Benefits of Resource Filtering | 227](#)
- [Types of Resource Filtering | 228](#)
- [Example Configurations from a Collector | 228](#)

On a fully loaded network device with thousands of IFLs and IFDs, getting data for all IFLs and IFDs can take several attempts and use a lot of bandwidth. Resource filtering lets a collector choose only the data it needs from a specific resource. This feature supports XPATH based resource filtering for AFT platforms and REGEX based filtering for PFE sensors on Junos microkernel.

Benefits of Resource Filtering

- You can select or reject specific instances of a resource, such as a key. This can include data from a particular IFL/IFD or all IFLs/IFDs.

- You can use a wildcard option to selectively choose resources for an element.

Types of Resource Filtering

- **XPATH-based filtering**- XPATH based filtering uses a key field or wildcard of one or more paths. The following options are supported in the XPATH based resource filtering:
 - Precise path or keys. Example: `/interfaces/interface[name=xe-0/0/0]`
 - Full wildcard of keys. Example: `/interfaces/interface[name=*]`
 - Partial wildcard of keys. Example: `/interfaces/interface[name=xe-0/0/*]`
 - Multi-level key options. i.e., data pertaining to a particular IFL or an IFD.
- **REGEX-based filtering**- REGEX filtering uses only a key field. There are many options of selecting a key, such as:
 - Precise path `xe-0/0/0`
 - Starts with `xe-0/*`
 - Ends with `*/0/0`
 - Not a particular interface `!xe-0/0/0`



NOTE:

- XPATH-based resource filtering is not applicable to UDP mode of transport.
- gNMI resource filtering and wild card filtering options are not applicable for firewall configuration path `/junos/system/linecard/firewall`, as there is no way to pass resource filter information in the path.

Example Configurations from a Collector

- **Precise path or key** such as `/interfaces/interface[name=xe-0/0/0]`:

An example configuration for a precise path or a key is given below for gRPC:

```
{
  "dut_list":
```

```
[
  {
    "ip": "ferrari-mx02",
    "port": 50051,
    "session_log": "grpc.session",
    "log_head": "grpc.data",
    "oc_rpc" : ["subscribe"],
    "subscribe": {
      "path_list": [{ "path":"/interfaces/
interface[name='et-0/0/0']", "filter": "", "sample_frequency": 2000, "suppress_unchanged": "",
"max_silent_interval": 0}],
      "input": {"collector_list": [{"address": "13.1.1.1"}]},
      "additional_config": {"limit_records": 1, "limit_time_seconds": 1}
    }
  }
]
```

- **Full wildcard of keys such as /interfaces/interface**

A sample configuration for a wildcard of keys is given below for gRPC:

```
{
  "dut_list":
  [
    {
      "ip": " ferrari-mx02",
      "port": 50051,
      "session_log": "grpc.session",
      "log_head": "grpc.data",
      "oc_rpc" : ["subscribe"],
      "subscribe": {
        "path_list": [{ "path":"/interfaces/interface/", "filter": "",
"sample_frequency": 2000, "suppress_unchanged": "", "max_silent_interval": 0}],
        "input": {"collector_list": [{"address": "13.1.1.1"}]},
        "additional_config": {"limit_records": 1, "limit_time_seconds": 1}
      }
    }
  ]
}
```

- **Partial wildcard of a key such as /interfaces/interface[name='et-0/0/*']**

A sample configuration for a partial wildcard of path is given below:

```
{
  "dut_list":
    [
      {
        "ip": "ferrari-mx02",
        "port": 50051,
        "session_log": "grpc.session",
        "log_head": "grpc.data",
        "oc_rpc" : ["subscribe"],
        "subscribe": {
          "path_list": [{ "path": "/interfaces/interface[name='et-0/0/*']"
        }, "filter": "",
        "sample_frequency": 2000, "suppress_unchanged": "", "max_silent_interval": 0}],
        "input": {"collector_list": [{"address": "13.1.1.1"}]},
        "additional_config": {"limit_records": 1, "limit_time_seconds": 1}
      }
    ]
}
```

- **gNMI Configuration**

A sample gNMI configuration is as shown below:

```
{
  "host": "gamoral-intf-mtt1-c",
  "user": "regress",
  "password": "MaRtInI",
  "port": 50051,
  "cid": "cid-45",
  "gnmi": {
    "mode": 0,
    "encoding": 2
  },
  "grpc": {
    "ws": 1048576
  },
  "paths": [
    {
      "path": "/interfaces/interface[name='et-0/0/31']/",

```

```

        "freq": 2000000000,
        "gnmi_submode": 2
    }
]
}

```

- **Firewall related such as a precise key filter:"__default_arp_policer__"**

As this is a Junos path, use the filter option in the Json file.

```

{
  "dut_list":
  [
    {
      "ip": "ferrari-mx01",
      "port": 50051,
      "session_log": "session.healthz_modify",
      "log_head": "data.ferrari_mx01",
      "oc_rpc" : ["subscribe"],
      "subscribe": {
        "path_list": [{ "path": "/junos/system/linecard/firewall/",
"filter": "__default_arp_policer__", "sample_frequency": 2000, "suppress_unchanged": "",
"max_silent_interval": 0}],
        "input": {"collector_list": [{"address": "10.213.2.16",
"port": 50051}]},
        "additional_config": {"limit_records": 1, "limit_time_seconds": 1,
"need_eos": 1}
      }
    }
  ]
}

```

- **Firewall related such as a partial wildcard key filter:"__default_arp_*"**

A sample configuration is as below:

```

{
  "dut_list":
  [
    {
      "ip": "ferrari-mx01",
      "port": 50051,

```

```

        "session_log": "session.healthz_modify",
        "log_head": "data.ferrari_mx01",
        "oc_rpc" : ["subscribe"],
        "subscribe": {
            "path_list": [{ "path": "/junos/system/linecard/firewall/",
"filter": "__default_arp_*", "sample_frequency": 2000, "suppress_unchanged": "",
"max_silent_interval": 0}],
            "input": {"collector_list": [{"address": "10.213.2.16",
"port": 50051}]},
            "additional_config": {"limit_records": 1, "limit_time_seconds": 1,
"need_eos": 1}
        }
    }
]
}

```

- **Firewall related such as no filter ---- filter:""**

In this case, data pertaining to all keys are exported.

```

{
  "dut_list":
  [
    {
      "ip": "ferrari-mx01",
      "port": 50051,
      "session_log": "session.healthz_modify",
      "log_head": "data.ferrari_mx01",
      "oc_rpc" : ["subscribe"],
      "subscribe": {
        "path_list": [{ "path": "/junos/system/linecard/firewall/",
"filter": "", "sample_frequency": 10000, "suppress_unchanged": "", "max_silent_interval": 0}],
        "input": {"collector_list": [{"address": "10.213.2.16",
"port": 50051}]},
        "additional_config": {"limit_records": 1, "limit_time_seconds": 1,
"need_eos": 1}
      }
    }
  ]
}

```




NOTE: If you add a new filter, an IFL, or an IFD and use a filter option in the telemetry subscription, the system applies the filter option to the newly added filter, IFL, or the IFD.

Transceiver Diagnostics

Junos OS Release 19.4R1 supports transceiver diagnostic sensors for ON_CHANGE and streaming statistics using JTI and gRPC services or gNMI services on MX960, MX2010, MX2020, PTX1000, PTX5000, and the PTX10000 line of routers. Use transceiver diagnostics to help troubleshoot problems, make decisions, and adjust resources as needed.

This feature supports OpenConfig transceiver model **openconfig-platform-transceiver.yang 0.5.0**.

Use the base resource path **/components/component/transceiver/** in a gRPC or gNMI subscription to export statistics from the Juniper device to an outside collector.

Fields that change continuously, such as temperature, input power, and output power, and laser bias current are not supported for ON_CHANGE.

VLAN Sensors

Junos OS Release 19.4R1 supports streaming VLAN statistics for ON_CHANGE using JTI and gRPC services on EX4650 and QFX5120 switches.

This feature supports OpenConfig model **openconfig-vlan.yang** configuration version 1.0.2.

Use the base resource path **/vlans/** in a gRPC subscription to export statistics from the Juniper device to an outside collector.

Other end points you can use in a subscription include:

- **/vlans/vlan/state/name**
- **/vlans/vlan/state/vlan-id**
- **/vlans/vlan/members/**
- **/vlans/vlan/members/member/interface-ref/state/interface/**

- /vlans/vlan/members/member/interface-ref/state/interface/switched-vlan/state/interface-mode
- /vlans/vlan/members/member/interface-ref/state/interface/switched-vlan/state/native-vlan
- /vlans/vlan/members/member/interface-ref/state/interface/switched-vlan/state/access-vlan
- /vlans/vlan/members/member/interface-ref/state/interface/switched-vlan/state/trunk-vlan
- /vlans/vlan/members/member/interface-ref/state/interface/vlan/state/vlan-id

5

PART

Best Practices for Implementing Junos Telemetry Interface

- [Telemetry Resources | 236](#)
 - [Guidelines for Specifying Data Reporting Intervals Junos Telemetry Interface | 237](#)
 - [Guidelines for Aggregating Junos Telemetry Interface Data | 238](#)
 - [Guidelines for Exporting Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets | 244](#)
-

Telemetry Resources

This section list all the resources required to deploy and manage the Junos Telemetry Interface in your network. The GitHub is a code hosting platform for version control and collaboration. Juniper Networks is part of the OpenConfig community that uses GitHub to develop telemetry code and store documents. Apart from GitHub resources Juniper also provides sensor explorer tools.

Table 24: Telemetry Resources

Telemetry Resources	Description
YANG models	Starting in Junos OS Evolved Release 23.4R1, all YANG data models for a given OS and release are in a single download package and GitHub repository. The package and repository include the native configuration, state, and RPC data models and the OpenConfig and Internet Engineering Task Force (IETF) data models supported by that OS.
Junos Telemetry Sensor Explorer	From releases 20.2R1 up to 23.1R1, the sensor information is available in the Telemetry Sensor Explorer.
"Legacy Sensor Paths" on page 255	Telemetry sensors and specific information about some legacy sensors.
Juniper telemetry on GitHub	Juniper telemetry models, augments, and deviations.
Protobuf file	Juniper protocol buffer files, organized by Junos OS Release.
gNMI protobuf file	Juniper gNMI protocol buffer files, organized by Junos OS Release.

Guidelines for Specifying Data Reporting Intervals

Junos Telemetry Interface

IN THIS SECTION

- [How to Determine the Reporting Interval for a System Resource | 237](#)

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:
 - $\text{Reporting interval} = \text{Required Export Interval Per Object} * \text{Minimum Number of objects reported on each Interval} / \text{Maximum Number of Objects}.$

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

Guidelines for Aggregating Junos Telemetry Interface Data

IN THIS SECTION

- [Aggregating Data Over Fixed Time Spans | 239](#)
- [Aggregating Data From Multiple Sources | 241](#)
- [Aggregating Data for Multiple Metrics | 243](#)

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" on page 239](#)
- ["Example: Aggregating Data for Cumulative Statistics" on page 240](#)

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 25 on page 239](#) for the specific values used in this example.

Table 25: 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 `JuniperNetworkSensors.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

Guidelines for Exporting Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets

You can use subscriber statistics and queue statistics for dynamic interfaces and interface-sets to support remote analytics and monitoring on MX Series routers that operate as a Broadband Network Gateway (BNG).

Before enabling export of subscriber statistics and queue statistics for dynamic interfaces and interface-sets, consider the following limitations:

- On MX Series routers supporting the Modular Port Concentrator 2 (MPC2), a slow internal refresh cycle for queue statistics can occur. This cycle can be lengthy at full line card scale. If the subscription frequency is higher than the internal refresh cycle, exported data may appear stale across reporting intervals.
- The unified in-service software upgrade (ISSU) feature enables you to upgrade your device between two different Junos OS releases with no disruption on the control plane and with minimal disruption of traffic. Dynamic interfaces and Interface-sets created prior to ISSU and prior to Junos OS Release 18.4R1 do not support telemetry for subscriber and queue statistics.
- The subscription frequency should be larger than the time to export telemetry. If the volume of data cannot be exported before the next reporting interval, the export continues to completion and the next reporting interval will immediately start. In such instances, continuous streaming results—behavior that may not be wanted.
- Multiple sensors from the dynamic-interfaces sub-tree may be subscribed to simultaneously. As streaming of these sensors for the sub-tree is supported by a single Junos component, you should expect the time to export the sensor data for each subscription to extend.
- Juniper advises to enable export only for active queues. To do this, include the `queues` statement at the `[[edit dynamic-profiles profile-name telemetryqueue-statistics $junos-interface-name]` or `[[edit dynamic-profiles profile-name telemetry queue-statistics $junos-interface-set-name]` hierarchy level. Exporting data for active queues only reduces the amount of data to export for each reporting interval.

RELATED DOCUMENTATION

Understanding Enabling Export of Subscriber Statistics and Queue Statistics for Dynamic Interfaces and Interface-Sets

Enable Export of Subscriber Statistics and Queue Statistics



Junos Telemetry Interface Plug-ins

- Network Telemetry Framework (NTF) Agent | **247**
 - Open Source Plug-ins | **251**
-

Network Telemetry Framework (NTF) Agent

IN THIS CHAPTER

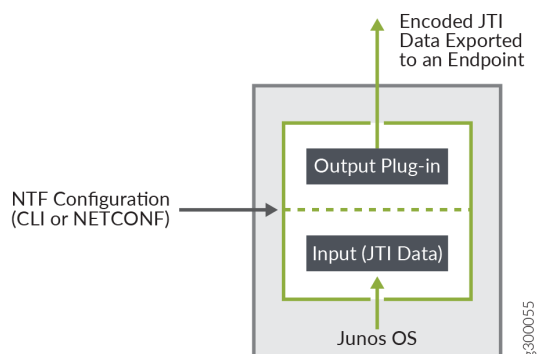
- [NTF Agent Overview | 247](#)
- [Configuring NTF Agent | 248](#)

NTF Agent Overview

Junos OS exposes telemetry data over gRPC and UDP as part of the Junos telemetry interface (JTI). One way to stream JTI data into your existing telemetry and analytics infrastructure requires managing an external entity to convert the data into a compatible format. Starting in Junos OS Release 18.4R1, the Network Telemetry Framework (NTF) agent feature provides an on-box solution that allows you to configure and customize to which endpoint (such as IPFIX and Kafka) the JTI data is delivered and in which format (such as AVRO, JSON, and MessagePack) the data is encoded.

NTF agent uses an output plug-in to translate JTI data into a format that is suitable for a particular endpoint. NTF agent subscribes to JTI data with user-defined sensor information. On receiving data, NTF agent uses the output plug-in to encode the data in the format that is required by the endpoint and then exports the translated data to the endpoint (see [Figure 9 on page 248](#)). NTF agent can be configured using Junos OS CLI or NETCONF.

Figure 9: NTF Agent Architecture



RELATED DOCUMENTATION

[Configuring NTF Agent](#) | 248

Configuring NTF Agent

To configure a Network Telemetry Framework (NTF) agent instance to send telemetry data to a single endpoint:

1. Create a service agent instance.

```
[edit services analytics agent]
user@host# edit service-agents agent-name
```

2. Configure parameters for the service agent input plug-in. The input plug-in options include analytics, input-ipfix, and input-jti-ipfix. See the *inputs* configuration statement for a description of the syntax.



NOTE: When you modify the input plug-in configuration of a service agent instance, the associated service agent daemon is restarted.

```
[edit services analytics agent service-agents agent-name]
user@host# edit inputs input-plugin-name parameters key-value-pairs
```


3. Configure parameters for the service agent output plug-in. Parameters are based on the key/value pair requirements of the output plug-in. For each service agent instance, you can configure only one endpoint to which to export data. The output plug-in options include `output-ipfix`, `kafka`, and `file`. See the `outputs` configuration statement for a description of the syntax.



NOTE: When you modify the output plug-in configuration of a service agent instance, the associated service agent daemon is restarted.

```
[edit services analytics agent service-agents agent-name]
user@host# set outputs output-plugin-name parameters key-value-pairs
```

4. (Optional) For each service agent instance, you can configure more than one input plug-in to push data to the output plug-in. To illustrate, the basic format of the configuration looks like:

```
[edit services analytics agent service-agents agent1]
  inputs {
    input-plugin1 {
      parameters {
        input-plugin1-key-value-pairs;
      }
    }
    input-plugin2 {
      parameters {
        input-plugin2-key-value-pairs;
      }
    }
  }
  outputs {
    output-plugin {
      parameters {
        output-plugin-key-value-pairs;
      }
    }
  }
```

5. (Optional) Delete a service agent instance.

```
user@host# delete services analytics agent service-agents agent-name
```

To configure tracing operations for NTF agent:

1. Specify the name of the file to receive the output of the tracing operation. The file is stored in the `/var/log/` directory of your device.

```
[edit services analytics agent]
user@host# edit traceoptions filename filename
```

2. Specify the severity level for messages to be logged.

```
[edit services analytics agent]
user@host# edit traceoptions flag {debug | error | info | trace}
```

SHOW COMMANDS and LOG FILES

1. Display the running service agent instances of the NTF agent.

```
user@host> show services analytics agent [brief | detail]
```

2. Additionally, view information about service agent instances, such as whether the input and output plug-ins have been initialized, in the service agent log file: `/var/log/agent-name.log`.

RELATED DOCUMENTATION

Configuring the BNG as an IPFIX Mediator to Collect and Export IPFIX Data

Configuring the Collection and Export of Local Telemetry Data on the IPFIX Mediator

IPFIX Mediation on the BNG

[NTF Agent Overview | 247](#)

Telemetry Data Collection on the IPFIX Mediator for Export to an IPFIX Collector

Open Source Plug-ins

IN THIS CHAPTER

- [JTI Plug-ins for Open Source Data Collectors | 251](#)

JTI Plug-ins for Open Source Data Collectors

Well-known open source data collectors, such as Telegraf have a plug-in-based architecture, where Junos Telemetry Interface (JTI) plug-ins can be written to translate JTI data into a format that can be easily understood by the collector. The following table provides links to the public JTI plug-in files for transporting JTI data over gNMI.

Table 26: JTI Plug-ins for Open Source Data Collectors

Open Source Data Collector	JTI Plug-ins for OpenConfig key-value Pairs over gNMI
Telegraf	jti_openconfig_telemetry_gnmi

7

PART

Configuration Statements and Operational Commands

- [Junos CLI Reference Overview | 253](#)
-

Junos CLI Reference Overview

We've consolidated all Junos CLI commands and configuration statements in one place. Read this guide to learn about the syntax and options that make up the statements and commands. Also understand the contexts in which you'll use these CLI elements in your network configurations and operations.

- [Junos CLI Reference](#)

Click the links to access Junos OS and Junos OS Evolved configuration statement and command summary topics.

- [Configuration Statements](#)
- [Operational Commands](#)

8

PART

Legacy Information

- [Legacy Sensor Paths | 255](#)
 - [J-Insight Device Monitor Overview | 335](#)
 - [J-Insight Device Monitor Basic Configuration | 338](#)
-

Legacy Sensor Paths

IN THIS SECTION

- Supported gRPC and gNMI Sensors | 255

Supported gRPC and gNMI Sensors

Starting with Junos OS Release 20.1R1, the on-device gRPC framework is upgraded to version v1.18.0 and is applicable to both JET and JTI. This version includes important enhancements for gRPC. Earlier legacy Junos OS platform versions (non-Occam) will continue to use version v1.3.0.

Starting with Junos OS Release 20.2R1, JTI supports MX routers with dual Routing Engines or MX Series Virtual Chassis on all Packet Forwarding Engine and Routing Engine sensors currently supported on MX Series routers. The level of sensor support currently available for MX Series routers applies, whether through streaming or ON_CHANGE statistics export, using UDP, remote procedure call (gRPC) services or gRPC Network Management Interface (gNMI) services. Additionally, JTI operational mode commands will provide details for all Routing Engines and MX Series Virtual Chassis, too.

Table 27: gRPC Sensors

Resource Path	Description
/interfaces/interface/state/forwarding-viable	Packet Forwarding Engine sensor for non-viable aggregated interface member links. This feature does not support non-LAG link members Starting in Junos OS Evolved Release 21.4R1, streaming statistics by means of gRPC and gNMI is supported on PTX10008 and PTX10016 routers.

Table 27: gRPC Sensors (Continued)

Resource Path	Description
<code>/junos/ike-security-associations/ike-security-association/routing-instance [name=' routing-instance-name]</code>	<p>Sensor for Internet Key Exchange (IKE) security statistics.</p> <p>When you configure a subscription request, use the reporting-interval parameter to configure the interval (in seconds) in which statistics are reported.</p> <p>Starting with Junos OS Release 18.1R1, MX Series routers are supported.</p> <ul style="list-style-type: none"> • remote-ip • local-ip • number-ipsec-sa-created • number-ipsec-sa-deleted • number-ipsec-sa-rekey • exchange-type • in-bytes • in-packets • out-bytes • out-packets • delete-payload-received • delete-payload-transmitted • dpd-request-payload-received • dpd-request-payload-transmitted • dpd-response-payload-received • dpd-response-payload-transmitted • dpd-response-payload-missed • dpd-response-payload-maximum-delay

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • dpd-response-seq-payload-missed • invalid-spi-notify-received • invalid-spi-notify-transmitted • routing-instance
<p>/junos/kernel/tcpip/rtstock</p>	<p>Sensor for kernel routing table socket (RTSOCK) information.</p> <p>Starting with Junos OS Release 19.3R1, EX9200, EX9251, EX9253, MX240, MX480, MX960, MX2010, MX2020, vMX, PTX1000, PTX10008, PTX10016, PTX3000 with RE-PTX-X8-64G, and PTX5000 with RE-PTX-X8-64G are supported.</p> <p>You can also add the following as the end path for /junos/kernel/rtsock/:</p> <ul style="list-style-type: none"> • total-error-cnt • total-veto-cnt

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/memory/	<p>Sensor for CPU memory. This sensor exports the CPU and memory utilization per process and CPU usage for threads per process. The current implementation is Linux-based; therefore, the export information and gathered output format differs significantly from this sensor's performance on previous platforms.</p> <p>Supported on MX Series routers with MPC10E-10C-MRATE and MPC10E-15C-MRATE line cards starting with Junos OS Release 19.3R1 for exporting telemetry information using gRPC services. This feature provides a different level of exported statistics in comparison to previous releases because it use the OpenConfig AFT model.</p> <p>Supported on MX2010 and MX2020 routers with MX2K-MPC11E line cards starting with Junos OS Release 20.1R1 for streaming telemetry information using gRPC services.</p> <p>Supported on EX2300, EX2300-MP, and EX3400 switches starting with Junos OS Release 20.2R1 and later for streaming telemetry information using gRPC services.</p> <p>Supported on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches starting with Junos OS Release 20.2R1 and later for INITIAL_SYNC statistics using gNMI services.</p> <p>The statistics exported from this sensor are found in the following operational mode commands: show system info, show system processes, and show system cpu.</p>
/junos/npu/memory	Starting with Junos OS Release 19.1R1, periodic streaming on QFX10002 switches and PTX10002 routers is supported.
/junos/services/health-monitor/config/	<p>Sensor for the health monitoring configuraiton.</p> <p>Starting with Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
<p>/junos/services/health-monitor/data/</p>	<p>Sensor for health monitoring data.</p> <p>Starting with Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p>
<p>/junos/services/ip-tunnel[name='tunnel-name']/usage/counters[name='counter-name']/</p>	<p>Sensor for Packet Forwarding Engine dynamic tunnels statistics.</p> <p>The statistics are used to report various network element performance metrics in a scalable and efficient way, providing visibility into Packet Forwarding Engine errors and drops.</p> <p>A timestamp indicating when the counters were last reset is included with all the exported data to allow collectors to determine if and when a reset event happened; for example, if the Packet Forwarding Engine hardware restarted.</p> <p>Exported statistics are similar to the output of the operational mode command <code>show nhdb hw dynamic-ip-tunnels</code>.</p> <p>Starting with Junos OS Release 17.4R1, MX Series devices are supported on gRPC services, with the exception of MX80 and MX104 routers. These routers support UDP export only for this sensor. To configure UDP export, include the sensor <code>/junos/services/ip-tunnel/usage/</code> in the sensor configuration statement at the <code>[edit services analytics]</code> hierarchy level.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/services/label-switched-path/usage/	<p>Sensor for LSP statistics. On MX Series routers only, the following are also supported: bidirectional LSPs for ultimate-hop popping (UHP).</p> <p>Starting with Junos OS Release 17.2R1, QFX10000 switches and PTX1000 routers are also supported.</p> <p>Starting with Junos OS Release 17.3R1, EX9200 switches are also supported.</p> <p>Starting with Junos OS Release 17.4R1 on MX Series and PTX Series routers only, statistics for bypass LSPs are also exported. Previously, only statistics for ingress LSPs were exported.</p> <p>Starting with Junos OS Release 18.2R1, QFX5100, QFX5110, and QFX5200 switches are also supported.</p> <p>Starting with Junos OS Release 18.3R1, QFX5120-48Y and EX4650 switches are also supported.</p> <p>Starting with Junos OS Release 18.4R1, EX4600 switches are also supported.</p> <p>Starting with Junos OS Release 19.1R1, PTX10001-20C routers support RSVP bypass LSPs originating at the transit node</p> <p>Starting with Junos OS Release 19.1R1, periodic streaming on QFX10002 switches and PTX10002 routers is supported.</p> <p>Starting in Junos OS Evolved Release 19.1R1, PTX10003 routers are supported.</p> <p>Starting with Junos OS Release 19.2R1, ACX6360 routers are supported.</p> <p>Starting with Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p> <p>Supported on QFX5200 switches starting with Junos OS Release 19.2R1 for streaming telemetry information using gNMI services.</p> <p>Starting with Junos OS Evolved Release 19.4R1, periodic streaming using gNMI services with PTX10003 routers is supported.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<p>Starting with Junos OS Evolved Release 20.2R1, periodic streaming using gRPC services with PTX10001 routers is supported.</p> <p>Periodic streaming is supported on QFX5120-48YM switches starting with Junos OS Release 20.4R1.</p> <p>For bypass LSPs, the following are exported:</p> <ul style="list-style-type: none"> • Bypass LSP originating at the ingress router of the protected LSP. • Bypass LSP originating at the transit router of the protected LSP. • Bypass LSP protecting the transit LSP as well as the locally originated LSP. <p>When the bypass LSP is active, traffic is exported both on the bypass LSP and the ingress (protected) LSP.</p> <p>NOTE: When you enable a sensor for LSP statistics only, you must also configure the sensor-based-stats (Junos Telemetry Interface) 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 statement or the enhanced-ethernet statement at the [edit chassis network-services] hierarchy level.</p>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
<div>/junos/services/segment-routing/interface/ingress/usage/</div> <div>/junos/services/segment-routing/interface/egress/usage/</div> <div>/junos/services/segment-routing/sid/usage/</div> <div>/junos/services/segment-routing/sid/egress/usage/</div>	<p>Sensors for aggregate segment routing traffic with IS-IS or OSPF.</p> <p>This sensor is supported on MX Series and PTX5000 routers starting with Junos OS Release 17.4R1.</p> <p>Starting with Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p> <p>Statistics are exported separately for each routing instance.</p> <p>The first path exports inbound traffic. The second path exports outbound traffic. The third path exports inbound segment routing traffic for each segment identifier.</p> <p>The path <code>/junos/services/segment-routing/sid/egress/usage/</code> can be used to collect SID statistics for traffic travelling towards the core.</p> <p>NOTE: When you enable a sensor for segment routing statistics, you must also configure the sensor-based-stats (Junos Telemetry Interface) statement at the [edit protocols isis source-packet-routing] or [edit protocols ospf source-packet-routing] hierarchy level.</p> <p>All MX and PTX5000 routers with FPC3 onwards support enhanced mode. If enhanced mode is not enabled, configure either the enhanced-ip statement or the enhanced-ethernet statement at the [edit chassis network-services] hierarchy level. On PTX Series routers, configure the enhanced-mode statement at the [edit chassis network-services] hierarchy level.</p> <p>NOTE: Currently, MPLS labels correspond only to only one instance, instance 0. Since each SID corresponds to a single instance_identifier, no aggregation is required to be done by the collector. The instance_identifier is stamped as 0.</p> <p>The following end points are supported:</p> <ul style="list-style-type: none"> <code>/network-instances/network-instance/mpls/signaling-protocols/segment-routing/interfaces/interface/state/in-pkts</code> <code>/network-instances/network-instance/mpls/signaling-protocols/segment-routing/interfaces/interface/state/in-octets</code>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /network-instances/network-instance/mpls/signaling-protocols/segment-routing/interfaces/interface/state/out-octets • /network-instances/network-instance/mpls/signaling-protocols/segment-routing/interfaces/interface/state/out-pkts • /network-instances/network-instance/mpls/aggregate-sid-counters/aggregate-sid-counter/state/in-octets • /network-instances/network-instance/mpls/aggregate-sid-counters/aggregate-sid-counter/state/in-pkts • /network-instances/network-instance/mpls/aggregate-sid-counters/aggregate-sid-counter/state/out-octets • /network-instances/network-instance/mpls/aggregate-sid-counters/aggregate-sid-counter/state/out-pkts • /network-instances/network-instance/mpls/interfaces/interface/sid-counters/sid-counter/state/in-octets • /network-instances/network-instance/mpls/interfaces/interface/sid-counters/sid-counter/state/in-pkts • /network-instances/network-instance/mpls/interfaces/interface/sid-counters/sid-counter/state/out-octets • /network-instances/network-instance/mpls/interfaces/interface/sid-counters/sid-counter/state/out-pkts • /network-instances/network-instance/mpls/interfaces/interface/sid-counters/sid-counter/forwarding-classes/forwarding-class/state/in-octets • /network-instances/network-instance/mpls/interfaces/interface/sid-counters/sid-counter/forwarding-classes/forwarding-class/state/in-pkts • /network-instances/network-instance/mpls/interfaces/interface/sid-counters/sid-counter/forwarding-classes/forwarding-class/state/out-octets

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> /network-instances/network-instance/mps/interfaces/interface/sid-counters/sid-counter/forwarding-classes/forwarding-class/state/out-pkts
/junos/services/segment-routing/sid/egress/usage/	Sensor for segment routing statistics based on segment identifier (SID).
/junos/services/segment-routing/sid/usage/	<p>Sensors for aggregate segment routing traffic with IS-IS.</p> <p>This sensor is supported on PTX3000 routers and PTX5000 routers with FPC2 starting with Junos OS Release 19.1R1.</p> <p>Statistics are exported separately for each routing instance.</p> <p>The first path exports inbound traffic. The second path exports outbound traffic. The third path exports inbound segment routing traffic for each segment identifier.</p> <p>NOTE: When you enable a sensor for segment routing statistics, you must also configure the sensor-based-stats (Junos Telemetry Interface) statement at the [edit protocols isis source-packet-routing] hierarchy level.</p>
/junos/services/segment-routing/traffic-engineering/ingress/usage	<p>Packet Forwarding Engine sensor for ingress segment routing traffic engineering (SR-TE) statistics.</p> <p>/junos</p> <p>Starting in Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p> <p>Starting in Junos OS Evolved Release 21.4R1, PTX10001-36MR, PTX10003, PTX10004, PTX10008, and PTX10016 routers are supported on gRPC and gNMI services.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
<p>/junos/services/segment-routing/traffic-engineering/transit/usage</p>	<p>Packet Forwarding Engine sensor for ingress segment routing traffic engineering statistics.</p> <p>Starting in Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p> <p>Starting in Junos OS Evolved Release 21.4R1,PTX10001-36MR, PTX10003, PTX10004, PTX10008,and PTX10016 routers are supported on gRPC and gNMI services.</p>
<p>/junos/services/segment-routing/traffic-engineering/tunnel/ingress/usage</p> <p>/junos/services/segment-routing/traffic-engineering/tunnel/transit/usage</p>	<p>Packet Forwarding Engine sensors for ingress and transit SR-TE statistics for uncolored LSPs.</p> <p>To configure SR-TE, see Configure SR-TE for Uncolored Ingress LSPs.</p>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
<p>/junos/services/segment-routing/traffic-engineering/tunnel/lsp/ingress/usage/</p> <p>/junos/services/segment-routing/traffic-engineering/tunnel/lsp/transit/usage/</p>	<p>Sensor for Segment Routing Traffic Engineering (SR-TE) per Label Switched Path (LSP) route statistics.</p> <p>You can stream SR-TE telemetry statistics for uncolored SR-TE policies to an outside collector. Ingress statistics include statistics for all traffic steered by means of an SR-TE LSP. Transit statistics include statistics for traffic to the Binding-SID (BSID) of the SR-TE policy.</p> <p>To enable these statistics, include the per-source per-segment-list option at the [edit protocols source-packet-routing telemetry statistics] hierarchy level.</p> <p>Starting in Junos OS Release 20.1R1, MX Series and PTX Series routers support streaming statistics using gRPC services.</p> <p>Starting in Junos OS Release 20.2R1, MX240, MX480, MX960, MX2010, and MX2020 with MPC-10E or MPC-11E routers support streaming statistics using gRPC services.</p> <p>When a subscription is made to these resource paths, the following output format is displayed:</p> <ul style="list-style-type: none"> • /mpls/signaling-protocols/segment-routing/sr-te-per-lsp-ingress-policies/sr-te-ingress-lsp-policy\[tunnel-name='srtelosp1' and source='st' and origin='0' and distinguisher='f' and lsp-name='sr1'\]/state/counters\[name='.*'\]/packets • /mpls/signaling-protocols/segment-routing/sr-te-per-lsp-transit-policies/sr-te-transit-lsp-policy\[tunnel-name='srtelosp1' and source='st' and origin='0' and distinguisher='f' and lsp-name='sr1'\]/state/counters\[name='.*'\]/packets <p>For the output format above, the field source, values can be ST (static tunnel) or PC (PCEP tunnel). For the field lsp-name, the value is the transit output. Other fields, such as Origin and Distinguisher are fixed for uncolored tunnels.</p>
/junos/system/cmerror/configuration	<p>Sensor for error monitoring configuration.</p> <p>Starting in Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/junos/system/cmerror/counters	<p>Sensor for error monitoring counters.</p> <p>Starting in Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p>
/junos/system/linecard/bmon-sw/	<p>Sensor for interface burst monitoring.</p> <p>Starting in Junos OS Evolved Release 19.3R1, QFX5220-128C and QFX5220-32CD switches are supported for streaming statistics on gRPC services.</p> <p>You can also add the following to the end of the path to stream specific statistics for interface burst monitoring:</p> <ul style="list-style-type: none"> • rx_bytes-Total number of bytes received during the export interval. • tx_bytes-Total number of bytes transmitted during the export interval. • start_ts-Start timestamp for the data collection window. • rx_peak_byte_rate-Maximum bytes rate per millisecond received from all the sampling intervals in the export interval. • rx_peak_ts-Timestamp of the first burst. • tx_peak_byte_rate-Maximum bytes rate per millisecond, transmitted from all the sampling intervals in the export interval. • tx_peak_byte_ts-Timestamp of the first transmit burst.
/junos/system/linecard/cos/interface/interface-set/output/queue/	<p>Sensor for logical interface (IFL)-set. The sensor streams queue statistics using Juniper proprietary gRPC and gNMI or by means of UDP. Zero suppression (suppressing zero values in statistics from streamed data) is also supported.</p> <p>Starting in Junos OS Release 22.3R1, MX204, MX240, MX304, MX480, MX960, MX2010, MX2020, MX10003, MX10004, MX10008, and MX10016 routers with Trio chipset EA, ZT, and YT-based fixed systems and modular systems linecards are supported.</p>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/junos/system/linecard/cpu/memory/	<p>Sensor for CPU memory.</p> <p>NOTE: On PTX Series routers, FPC1 and FPC2 are not supported.</p> <p>Supported on QFX10000 switches and PTX1000 routers starting with Junos OS Release 17.2R1.</p> <p>Supported on EX9200 switches starting with Junos OS Release 17.3R1.</p> <p>Supported on QFX5100, QFX5110, and QFX5200 switches starting with Junos OS Release 18.2R1.</p> <p>Supported on QFX5120-48Y and EX4650 switches starting with Junos OS Release 18.3R1.</p> <p>Supported on EX4600 switches starting with Junos OS Release 18.4R1.</p> <p>Periodic streaming is supported on on QFX10002 switches and PTX10002 routers starting with Junos OS Release 19.1R1.</p> <p>Starting with Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p> <p>Supported on QFX5200 switches starting with Junos OS Release 19.2R1 for streaming telemetry information using gNMI services.</p> <p>Periodic streaming using gRPC services is supported on EX4300-MP switches starting with Junos OS Release 19.4R1,</p> <p>Periodic streaming using gRPC services is supported on EX2300, EX2300-MP, and EX3400 switches starting with Junos OS Release 20.2R1.</p> <p>Periodic streaming is supported on QFX5120-48YM switches starting with Junos OS Release 20.4R1.</p> <p>Streaming statistics using Juniper proprietary gRPC is supported on M304 starting with Junos OS Release 22.4R1.</p> <p>You can also include the following to end of the resource path for CPU memory:</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<ul style="list-style-type: none">• [name="mem-util-<memory-name>-size"]/value• [name="mem-util-<memory-name>-bytes-allocated"]/value• [name="mem-util-<memory-name>-utilization"]/value• [name="mem-util-<memory-name>-< app-name>-allocations"]/value• [name="mem-util-<memory-name>-< app-name>-frees"]/value• [name="mem-util-<memory-name>-< app-name>-allocations-failed"]/value

Table 27: gRPC Sensors (Continued)

Resource Path	Description
<code>/junos/system/linecard/ddos/</code>	<p>Distributed denial of service (DDoS) sensor. This sensor supports the Openconfig data model <code>junos/ui/openconfig/yang/</code> and <code>junos-ddos.yang</code>.</p> <p>You can stream information using Juniper proprietary gRPC or UDP (native) export.</p> <p>There are 45 packet types for DDoS. To maintain a reasonably sized data stream, data is exported for all protocols that have seen traffic using the zero-suppression model.</p> <p>On QFX5000 platforms, multiple protocols can share the same CPU queue. DDoS configurations are applied at the CPU queue level. Consequently, DDoS statistics fetched from the CPU queue will return the aggregate value of all protocols using that queue. For example, if BGP, LDP, and RSVP protocols are using a particular CPU queue, but the DDoS limit is violated only by the BGP protocol, the DDoS violation reported will include all three protocols: BGP, LDP, and RSVP. This information will be exported to the collector with the DDoS sensor.</p> <p>Starting in Junos OS Release 22.1R1 EX4650, QFX5110, QFX5120-48Y, QFX5200 and QFX5210 switches are supported.</p> <p>Starting in Junos OS Evolved Release 22.3R1, PTX10001-36MR, PTX10003, PTX10004, PTX10008, PTX10016 routers are supported.</p> <p>You can also add the following leaves to the end of the path to stream specific statistics:</p> <ul style="list-style-type: none"> • <code>group_name</code> • <code>group_id</code> • <code>protocol_name</code> • <code>protocol_id</code> • <code>location</code> • <code>received</code>

Table 27: gRPC Sensors (Continued)

Resource Path	Description																																																																								
	<ul style="list-style-type: none">• arrive-policer• dropped-individual_policer• dropped_aggregate_policer• total_dropped• final_passed• arrival_rate• max_arrival_rate• pass_rate• policer_state• policer_violation_count• policer_violation_start_time• policer_violation_end_time• policer_violation_duration <p>The following packet types are supported:</p> <table><tr><th>CMICQ</th><th>Channel</th><th>bwidth</th><th>burst</th><th>Qlen</th><th>Proto(s)</th></tr><tr><td>0</td><td>3</td><td>500</td><td>10</td><td>200</td><td>uncls</td></tr><tr><td>4</td><td>1</td><td>4000</td><td>200</td><td>200</td><td>vchassis</td></tr><tr><td>7</td><td>3</td><td>500</td><td>200</td><td>200</td><td>vxlan</td></tr><tr><td>8</td><td>3</td><td>1500</td><td>200</td><td>200</td><td>localnh</td></tr><tr><td>9</td><td>3</td><td>1000</td><td>200</td><td>200</td><td>vcipc-udp</td></tr><tr><td>10</td><td>3</td><td>2000</td><td>200</td><td>200</td><td>sample-source</td></tr><tr><td>11</td><td>3</td><td>2000</td><td>200</td><td>200</td><td>sample-dest</td></tr><tr><td>12</td><td>3</td><td>50</td><td>10</td><td>200</td><td>l3mtu-</td></tr><tr><td colspan="6">fail,ttl,ip-opt.</td></tr><tr><td>14</td><td>3</td><td>100</td><td>10</td><td>200</td><td>garp-reply</td></tr><tr><td>15</td><td>3</td><td>500</td><td>10</td><td>200</td><td>fw-host</td></tr></table>	CMICQ	Channel	bwidth	burst	Qlen	Proto(s)	0	3	500	10	200	uncls	4	1	4000	200	200	vchassis	7	3	500	200	200	vxlan	8	3	1500	200	200	localnh	9	3	1000	200	200	vcipc-udp	10	3	2000	200	200	sample-source	11	3	2000	200	200	sample-dest	12	3	50	10	200	l3mtu-	fail,ttl,ip-opt.						14	3	100	10	200	garp-reply	15	3	500	10	200	fw-host
CMICQ	Channel	bwidth	burst	Qlen	Proto(s)																																																																				
0	3	500	10	200	uncls																																																																				
4	1	4000	200	200	vchassis																																																																				
7	3	500	200	200	vxlan																																																																				
8	3	1500	200	200	localnh																																																																				
9	3	1000	200	200	vcipc-udp																																																																				
10	3	2000	200	200	sample-source																																																																				
11	3	2000	200	200	sample-dest																																																																				
12	3	50	10	200	l3mtu-																																																																				
fail,ttl,ip-opt.																																																																									
14	3	100	10	200	garp-reply																																																																				
15	3	500	10	200	fw-host																																																																				

Table 27: gRPC Sensors (Continued)

Resource Path	Description					
	16	3	500	200	200	ndpv6
	17	3	1000	200	200	dhcqv4v6
	19	3	1500	200	200	ipmc-reserved
	20	3	300	200	200	resolve
	21	3	100	10	200	l3dest-miss
	22	3	100	10	200	redirect
	23	3	300	200	200	l3nhop
	24	3	100	10	200	l3mc-sgwhit-icl
	25	3	50	10	200	martian-address
	26	3	1000	200	200	l2pt
	27	3	50	10	200	urpf-fail
	28	3	1000	300	300	ipmcast-miss
	29	2	300	10	200	nonucast-switch
	30	2	3000	200	200	
	rsvp,ldp,bgp					
	31	2	3000	200	200	unknown-
	l2mc,rip,ospf					
	32	2	1000	200	200	fip-snooping
	33	2	1000	200	200	igmp
	34	2	500	200	200	arp
	35	2	1500	200	200	pim-data
	36	2	1500	200	200	ospf-hello
	37	2	1500	200	200	pim-ctrl
	38	2	2000	200	200	isis
	39	1	250	200	200	lACP
	40	1	1200	200	200	bfd
	41	1	100	10	200	ntp
	42	1	500	200	200	vchassis
	43	1	1000	200	200	
	stp,pvstp,lldp					

Table 27: gRPC Sensors (Continued)

Resource Path	Description
/junos/system/linecard/environment/	<p>Sensor for environmental statistics.</p> <p>When subscribing to the resource path /junos/system/linecard/environment, the prefix for the streamed path at the collector side was displaying as /junos/linecard/environment. This issue is resolved in Junos OS 23.1R1 and Junos OS Evolved 23.1R1 and the subscription path and the streamed path match to display /junos/system/linecard/environment.</p> <p>Supported on MX10008 routers starting with Junos OS Release 21.4R1 using Juniper proprietary gRPC.</p> <p>Supported on MX10004 routers starting with Junos OS Release 22.3R1 using Juniper proprietary gRPC.</p> <p>FPC environment sensor /junos/system/linecard[name=FPC0]/environment/ is supported on MX10004 routers starting with Junos OS Release 22.4R1 using Juniper proprietary gRPC and gNMI. You can stream the following endpoints:</p> <ul style="list-style-type: none"> • /power-record/max-fpc-power • /power-record/fpc-power • voltage-record/voltage-sensor-name • voltage-record/voltage-value • temp-sensor-name • temp-record/temp-value

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/system/linecard/fabric/	<p>Sensor for fabric statistics.</p> <p>Supported on MX10008 routers starting with Junos OS Release 22.1R1 using Juniper proprietary gRPC.</p> <p>Supported on MX10004 routers starting with Junos OS Release 22.4R1 using Juniper proprietary gRPC.</p> <p>Supported on PTX10001-36MR, PTX10003, PTX10004, PTX10008, and PTX10016 routers starting with Junos OS Evolved Release 22.4R1 using Juniper proprietary gRPC.</p> <p>Subscribe to this resource path to export the following statistics under the base path /junos/fabric-statistics/fabric-message/edges/class-stats/transmit-counts/:</p> <ul style="list-style-type: none"> • packets • bytes • packets-per-second • bytes-per-second • drop-packets • drop-bytes • drop-packets-per-second • drop-bytes-per-second • queue-depth-average • queue-depth-current • queue-depth-peak • queue-depth-maximum • error-packets • error-packets-per-second

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	Supported on MX10008 routers starting with Junos OS Release 21.4R1 using Juniper proprietary gRPC.

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/system/linecard/firewall/	<p>Sensor for firewall filter counters and policer counters. Each line card reports counters separately.</p> <p>Supported on QFX10000 switches starting with Junos OS Release 17.2R1.</p> <p>Supported on PTX1000 routers and EX9200 switches starting with Junos OS Release 17.3R1.</p> <p>Supported on QFX5100, QFX5110, and QFX5200 switches starting with Junos OS Release 18.2R1.</p> <p>Supported on QFX5120-48Y and EX4650 switches starting with Junos OS Release 18.3R1.</p> <p>Supported on EX4600 switches starting with Junos OS Release 18.4R1.</p> <p>Starting with Junos OS Release 19.1R1, periodic streaming is supported on QFX10002 switches and PTX10002 routers.</p> <p>Starting in Junos OS Evolved Release 19.1R1, PTX10003 routers are supported.</p> <p>Starting in Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p> <p>Supported on QFX5200 switches starting with Junos OS Release 19.2R1 for streaming telemetry information using gNMI services.</p> <p>Supported on MX240, MX480, and MX960 routers starting with Junos OS Release 19.3R1 for exporting telemetry information using gNMI services. This feature includes support to export telemetry data for integration with AFTTelemetry and LibTelemetry libraries with the OpenConfig model openconfig-aft.</p> <p>Periodic streaming using gRPC services with EX4300-MP switches is supported starting with Junos OS Release 19.4R1.</p> <p>Periodic streaming using gNMI services with PTX10003 routers is supported starting with Junos OS Evolved Release 19.4R1.</p> <p>Periodic streaming using gNMI services on MX2K-MPC11E line cards on MX2010 and MX2020 routers is supported starting with Junos OS Release 20.1R1.</p>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
	<p>Periodic streaming using gRPC services is supported on EX2300, EX2300-MP, and EX3400 switches starting with Junos OS Release 20.2R1.</p> <p>INITIAL_SYNC statistics using gNMI services is supported on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches starting with Junos OS Release 20.2R1.</p> <p>Periodic streaming is supported on QFX5120-48YM switches starting with Junos OS Release 20.4R1.</p> <p>NOTE: Hierarchical policer statistics are collected for MX Series routers only. Traffic-class counter statistics are collected for PTX Series routers and QFX10000 switches only.</p> <p>Firewall counters are exported even if the interface to which the firewall filter is attached is operationally down.</p> <p>The following OpenConfig paths are supported:</p> <ul style="list-style-type: none"> • junos/firewall[name='filter-name']/timestamp • /junos/firewall[name='filter-name']/ memory-usage/[name='memory-type']/allocated • /junos/firewall[name='filter-name']/ counter/[name='counter-name']/packets
/junos/system/linecard/intf-exp/	<p>Interface express sensor.</p> <p>This sensor leverages statistics out of the physical interface sensor, providing faster and more frequent operational status statistics. Only the physical interfaces' operational status from the Flexible PIC Concentrator (FPC) is collected and reported. Statistics from the Routing Engine interface are not reported.</p> <p>Supported on PTX1000, PTX3000, PTX5000, and PTX10000 starting with Junos OS Release 18.1R1.</p> <p>Supported on MX960, MX2010, and MX2020 routers starting with Junos OS Release 19.3R1.</p>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/junos/system/linecard/interface/	<p>Packet Forwarding Engine sensor for physical interface traffic.</p> <p>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 the slot on which an interface is configured.</p> <p>For Aggregated Ethernet interfaces, statistics are exported for the member physical interfaces. You must aggregate the counters at the destination server, or collector.</p> <p>If a physical interface is administratively down or operationally down, interface counters are not exported.</p> <p>Issuing an operational clear command, such as <code>clear interfaces statistics all</code>, does not reset statistics exported by the line card.</p> <p>Supported on PTX Series routers starting with Junos OS Release 15.1F3. Supported on MX Series routers starting with Junos OS Release 15.1F5.</p> <p>Supported on QFX10000 switches and PTX1000 routers starting with Junos OS Release 17.2R1.</p> <p>Supported on EX9200 switches and MX150 routers starting with Junos OS Release 17.3R1.</p> <p>Supported on QFX5100, QFX5110, and QFX5200 switches starting with Junos OS Release 18.2R1.</p> <p>Supported on QFX5120-48Y and EX4650 switches starting with Junos OS Release 18.3R1.</p> <p>Supported on EX4600 switches Starting with Junos OS Release 18.4R1.</p> <p>Periodic streaming is supported on QFX10002 switches and PTX10002 routers starting with Junos OS Release 19.1R1.</p> <p>Supported on MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches with Junos OS Release 19.2R1 on gRPC and gNMI services.</p> <p>Supported on MX240, MX480, and MX960 routers starting with Junos OS Release 19.3R1 for exporting telemetry information using gNMI services. This feature includes support to export telemetry</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<p>data for integration with AFTTelemetry and LibTelemetry libraries with the OpenConfig model openconfig-aft.</p> <p>Starting with Junos OS Release 19.4R1, periodic streaming using gRPC services with EX4300-MP switches is supported.</p> <p>Periodic streaming using gNMI services on MX2K-MPC11E line cards on MX2010 and MX2020 routers is supported starting with Junos OS Release 20.1R1.</p> <p>Periodic streaming using gRPC services is supported on EX2300, EX2300-MP, and EX3400 switches starting with Junos OS Release 20.2R1.</p> <p>INITIAL_SYNC statistics using gNMI services is supported on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches starting with Junos OS Release 21.4R1.</p> <p>Streaming statistics using gRPC services or gNMI services is supported on PTX10008 routers starting with Junos OS Release 20.2R1.</p> <p>Periodic streaming is supported on QFX5120-48YM switches starting with Junos OS Release 20.4R1.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/system/linecard/interface/logical/family/ipv4/usage/	Sensor for per-family logical interface input and output counters for IPv4 and IPv6 traffic.
/junos/system/linecard/interface/logical/family/ipv6/usage/	<p>Streaming statistics using Juniper proprietary gRPC is supported on MX304 starting with Junos OS Release 22.4R1.</p> <p>Streaming statistics using Juniper proprietary gRPC is supported on MX10008 starting with Junos OS Release 21.4R1.</p> <p>Streaming statistics using Juniper proprietary gRPC is supported on MX Series and PTX Series routers using third-generation FPCs starting with Junos OS Release 21.2R1.</p> <p>Streaming statistics using Juniper proprietary gRPC is supported on MX10004 starting with Junos OS Release 22.3R1.</p> <p>Streaming of IPv6 counters supported on PTX10001-36MR, PTX10004, PTX10008 and PTX10016 routers starting with Junos OS Evolved Release 23.2R1.</p>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/junos/system/linecard/interface/logical/usage	<p>Packet Forwarding Engine sensor for logical interface statistics.</p> <p>NOTE: If a logical interface is operationally down, interface statistics continue to be exported.</p> <p>Issuing an operational clear command, such as <code>clear interfaces statistics all</code>, does not reset statistics exported by the line card.</p> <p>NOTE: If a logical interface is operationally down, interface statistics continue to be exported.</p> <p>Issuing an operational clear command, such as <code>clear interfaces statistics all</code>, does not reset statistics exported by the line card.</p> <p>NOTE: Locally injected packets from the Routing Engine are not exported.</p> <p>NOTE: Locally injected packets from the Routing Engine are not exported.</p> <p>Supported in Junos OS Release 15.1F5.</p> <p>Supported QFX10000 switches starting with on Junos OS Release 17.2R1.</p> <p>Supported on EX9200 switches and MX150 routers starting with Junos OS Release 17.3R1.</p> <p>Supported on QFX5100, QFX5110, and QFX5200 switches starting with Junos OS Release 18.2R1.</p> <p>Supported on QFX5120-48Y and EX4650 switches starting with Junos OS Release 18.3R1.</p> <p>Supported on EX4600 switches starting with Junos OS Release 18.4R1.</p> <p>Starting with Junos OS Release 19.1R1, periodic streaming is supported on QFX10002 switches and PTX10002 routers.</p> <p>Supported on MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches with Junos OS Release 19.2R1 on gRPC and gNMI services.</p> <p>Supported on QFX5200 switches starting with Junos OS Release 19.2R1 for streaming telemetry information using gNMI services.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<p>Supported on MX240, MX480, and MX960 routers starting with Junos OS Release 19.3R1 for exporting telemetry information using gNMI services. This feature includes support to export telemetry data for integration with AFTTelemetry and LibTelemetry libraries with the OpenConfig model openconfig-aft.</p> <p>Starting with Junos OS Release 19.4R1, periodic streaming using gRPC services with EX4300-MP switches is supported.</p> <p>Periodic streaming using gNMI services on MX2K-MPC11E line cards on MX2010 and MX2020 routers is supported starting with Junos OS Release 20.1R1.</p> <p>Periodic streaming using gRPC services is supported on EX3400 switches starting with Junos OS Release 20.2R1.</p> <p>INITIAL_SYNC statistics using gNMI services is supported on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches starting with Junos OS Release 20.2R1.</p> <p>Periodic streaming is supported on QFX5120-48YM switches starting with Junos OS Release 20.4R1.</p> <p>Supported on PTX10003 routers starting in Junos OS Evolved Release 22.3R1. Support includes the following sensors:</p> <ul style="list-style-type: none"> • /junos/system/linecard/interface/logical/usage/counters/in-pkts/state/value • /junos/system/linecard/interface/logical/usage/counters/out-pkts/state/value • /junos/system/linecard/interface/logical/usage/counters/in-octets/state/value • /junos/system/linecard/interface/logical/usage/counters/out-octets/state/value <p>Supported on virtual interfaces (lt, gr, si, lsp and ps) on MX204, MX480, MX960, MX10004, MX10008, MX10016, MX2010, and MX2020 routers starting in Junos OS Release 23.2R1.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/system/linecard/interface/queue/	<p>Sensor for interface queue statistics.</p> <p>Starting with Junos OS Release 18.3R1, when a subscription is made to /interfaces on MX, EX, QFX, PTX, and ACX7k platforms that support this feature, traffic and queue statistics are delivered in two separate sensors:</p> <ul style="list-style-type: none"> • /junos/system/linecard/interface/traffic/ • /junos/system/linecard/interface/queue/ <p>This can reduce the reap time for non-queue data for platforms supporting Virtual Output Queues (VOQ), such as PTX Series routers.</p> <p>Starting in Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p> <p>Supported on MX240, MX480, and MX960 routers starting with Junos OS Release 19.3R1 for exporting telemetry information using gNMI services. This feature includes support to export telemetry data for integration with AFTTelemetry and LibTelemetry libraries with the OpenConfig model openconfig-aft.</p> <p>Periodic streaming using gNMI services on MX2K-MPC11E line cards on MX2010 and MX2020 routers is supported starting with Junos OS Release 20.1R1.</p> <p>INITIAL_SYNC statistics using gNMI services is supported on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches starting with Junos OS Release 20.2R1.</p> <p>Supported on ACX7100 and ACX7509 starting in Junos OS Evolved Release 22.2R1. Support includes transmitted counters for:</p> <ul style="list-style-type: none"> • packets • bytes • red drop packets • red drop bytes

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
	<ul style="list-style-type: none"> • tail drop packets • tail drop bytes <p>Starting in Junos OS Evolved Release 22.4R1 you can stream statistics for IPv4 and IPv6 traffic statistics using the resource path / junos/system/linecard/interface/traffic/. Support includes transmitted counters for:</p> <ul style="list-style-type: none"> • if_in_ipv4pkts • if_out_ipv4pkts • if_in_ipv6pkts • if_out_ipv6pkts <p>Supported on virtual interfaces (lt, gr, si, lsp and ps) on MX204, MX480, MX960, MX10004, MX10008, MX10016, MX2010, and MX2020 routers starting in Junos OS Release 23.2R1.</p>
/junos/system/linecard/node-slicing/af-fab-stats/	<p>Sensor to export abstracted fabric (AF) interface-specific load-balancing and fabric queue statistics. This sensor is only supported for in node virtualization configurations on MX routers with an AF Interface as the connecting link between guest network functions (GNFs).The sensor also reports aggregated statistics across all AF interfaces hosted on a source packet forwarding engine of local guest GNFs along with the fabric statistics for all traffic ingressing from and egressing to the fabric from that the packet forwarding engine.</p> <p>Supported on MX480, MX960, MX2010, MX2020, MX2008 and MX-ELM routers with Junos OS Release 18.4R1.</p>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
/junos/system/linecard/npu/memory/	<p>Sensor for network processing unit (NPU) memory.</p> <p>You can also add the following leaves to the end of the path to stream specific statistics:</p> <ul style="list-style-type: none"> • resource_name • size • allocated • utilization • lower-threshold • upper-threshold • health <p>NOTE: Collecting telemetry statistics using the NPU memory sensor can cause high CPU cycles when gathering the MPLS ingress statistics (MPLS_Entry leaf). This, in turn, creates performance issues for the packet forwarding engine process. To correct this problem, MPLS ingress statistics are initially collected as a baseline. Statistics are updated after 30 seconds if a route is added or deleted. Otherwise, if there is no route change, statistics are fetched every hour.</p> <p>Supported on EX4650, QFX5110, QFX5120-48Y, QFX5200, and QFX5210 switches starting with Junos OS Release 21.4R1 for exporting telemetry information using gRPC or gNMI services.</p> <p>Supported on QFX5130, QFX5700, QFX5220-32CD, QFX5220-128C, QFX5130-48C, QFX5130-48CM, QFX5130E-32CD, QFX5230-64CD, QFX5240-64QD, and QFX5240-64QD switches starting with Junos OS Release 24.4R1 for exporting telemetry information using gRPC or gNMI services.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
<code>/junos/system/linecard/npu/memory/</code>	<p>Sensor for network processing unit (NPU) memory.</p> <p>Supported on MX Series routers with MPC10E-10C-MRATE and MPC10E-15C-MRATE line cards starting with Junos OS Release 19.3R1 for exporting telemetry information using gRPC services. This feature provides a different level of exported statistics in comparison to previous releases because it use the OpenConfig AFT model.</p> <p>Supported on MX2010 and MX2020 routers with MX2K-MPC11E line cards starting with Junos OS Release 20.1R1 for streaming telemetry information using gRPC services.</p> <p>Supported on MX304 routers using Juniper proprietary gRPC and gNMI starting with Junos OS Release 22.4R1.</p> <p>You can also add the following to the end of the path to stream specific statistics for NPU memory:</p> <ul style="list-style-type: none"> • <code>mem-util-edmem-size</code> • <code>mem-util-edmem-allocated</code> • <code>mem-util-edmem-utilization</code> • <code>mem-util-idmem-size</code> • <code>mem-util-idmem-allocated</code> • <code>mem-util-idmem-utilization</code> • <code>mem-util-bulk-dmem-size</code> • <code>mem-util-bulk-dmem-allocated</code> • <code>mem-util-bulk-dmem-utilization</code> • <code>mem-util-next-hop-edmem-size</code> • <code>mem-util-next-hop-edmem-allocated</code> • <code>mem-util-next-hop-edmem-utilization</code> • <code>mem-util-next-hop-bulk-dmem-size</code>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • mem-util-next-hop-bulk-dmem-allocated • mem-util-next-hop-bulk-dmem-utilization • mem-util-next-hop-idmem-size • mem-util-next-hop-idmem-allocated • mem-util-next-hop-inline-services-free-count • mem-util-next-hop-mobile:-timing-profile-bytes-allocated • mem-util-next-hop-mobile:-timing-profile-allocation-count • mem-util-next-hop-mobile:-timing-profile-free-count • mem-util-next-hop-packet-reassembly-(rw)-bytes-allocated • mem-util-next-hop-packet-reassembly-(rw)-allocation-count • mem-util-next-hop-packet-reassembly-(rw)-free-count • mem-util-next-hop-packet-reassembly---persistent-(rw)-bytes-allocated • mem-util-next-hop-packet-reassembly---persistent-(rw)-allocation-count • mem-util-next-hop-packet-reassembly---persistent-(rw)-free-count • mem-util-next-hop-ml-bundle-bytes-allocated • mem-util-next-hop-ml-bundle-allocation-count • mem-util-next-hop-ml-bundle-free-count • mem-util-next-hop-ddos-scf-d-params-bytes-allocated • mem-util-next-hop-ddos-scf-d-params-allocation-count • mem-util-next-hop-ddos-scf-d-params-free-count • mem-util-next-hop-vbf-bytes-allocated

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<ul style="list-style-type: none"> • mem-util-next-hop-vbf-allocation-count • mem-util-next-hop-vbf-free-count • mem-util-next-hop-ptp-ieee-1588-nhs-bytes-allocated • mem-util-next-hop-ptp-ieee-1588-nhs-allocation-count • mem-util-next-hop-ptp-ieee-1588-nhs-free-count • mem-util-next-hop-cos-bytes-allocated • mem-util-next-hop-cos-allocation-count • mem-util-next-hop-cos-free-count • mem-util-next-hop-inline-hash-sessions-bytes-allocated • mem-util-next-hop-inline-hash-sessions-allocation-count • mem-util-next-hop-inline-hash-sessions-free-count • mem-util-next-hop-inline-mdi-bytes-allocated • mem-util-next-hop-inline-mdi-allocation-count • mem-util-next-hop-inline-mdi-free-count • mem-util-next-hop-cos-enhanced-priority-bytes-allocated • mem-util-next-hop-cos-enhanced-priority-allocation-count • mem-util-next-hop-cos-enhanced-priority-free-count • mem-util-firewall-fw-bytes-allocated • mem-util-firewall-fw-allocation-count • mem-util-firewall-fw-free-count • mem-util-counters-fw-counter-bytes-allocated • mem-util-counters-fw-counter-allocation-count

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • mem-util-counters-fw-counter-free-count • mem-util-counters-fw-policer-bytes-allocated • mem-util-counters-fw-policer-allocation-count • mem-util-counters-fw-policer-free-count • mem-util-counters-ifd-error-cntr-bytes-allocated • mem-util-counters-ifd-error-cntr-allocation-count • mem-util-counters-ifd-error-cntr-free-count • mem-util-counters-nh-cntr-bytes-allocated • mem-util-counters-nh-cntr-allocation-count • mem-util-counters-nh-cntr-free-count • mem-util-counters-ifl-cntr-bytes-allocated • mem-util-counters-ifl-cntr-allocation-count • mem-util-counters-ifl-cntr-free-count • mem-util-counters-bridge-domain-counter0-bytes-allocated • mem-util-counters-bridge-domain-counter0-allocation-count • mem-util-counters-bridge-domain-counter0-free-count • mem-util-counters-bridge-domain-counter0-free-count • mem-util-counters-bridge-domain-cntr-bytes-allocated • mem-util-counters-bridge-domain-cntr-allocation-count • mem-util-counters-bridge-domain-cntr-free-count • mem-util-counters-sample-inline-params-bytes-allocated • mem-util-counters-sample-inline-params-allocation-count

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • mem-util-counters-sample-inline-params-free-count • mem-util-counters-services-counters-bytes-allocated • mem-util-counters-services-counters-allocation-count • mem-util-counters-services-counters-free-count • mem-util-counters-exception-counter-bytes-allocated • mem-util-counters-exception-counter-allocation-count • mem-util-counters-exception-counter-free-count • mem-util-counters-issu-policer-bytes-allocated • mem-util-counters-issu-policer-allocation-count • mem-util-counters-issu-policer-free-count • mem-util-counters-ddos-scf-d-counters-bytes-allocated • mem-util-counters-ddos-scf-d-counters-allocation-count • mem-util-counters-ddos-scf-d-counters-free-count • mem-util-counters-ip-reassembly-counter-bytes-allocated • mem-util-counters-ip-reassembly-counter-allocation-count • mem-util-counters-ip-reassembly-counter-free-count • mem-util-hash-hash-edmem-overhead-bytes-allocated • mem-util-hash-hash-edmem-overhead-bytes-allocated • mem-util-hash-hash-edmem-overhead-bytes-allocated • mem-util-hash-hash-edmem-overhead-bytes-allocated • mem-util-hash-hash-edmem-overhead-bytes-allocated • mem-util-hash-hash-edmem-overhead-allocation-count

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<ul style="list-style-type: none"> • mem-util-hash-hash-edmem-overhead-free-count • mem-util-hash-hash-edmem-bkt-bytes-allocated • mem-util-hash-hash-edmem-bkt-allocation-count • mem-util-hash-hash-edmem-bkt-free-count • mem-util-hash-hash-edmem-rec-bytes-allocated • mem-util-hash-hash-edmem-rec-allocation-count • mem-util-hash-hash-edmem-rec-free-count • mem-util-hash-hash-edmem-sideband-bytes-allocated • mem-util-hash-hash-edmem-sideband-allocation-count • mem-util-hash-hash-edmem-sideband-free-count • mem-util-hash-hash-dmem-bkt-bytes-allocated • mem-util-hash-hash-dmem-bkt-allocation-count • mem-util-hash-hash-dmem-bkt-free-count • mem-util-hash-hash-dmem-rec-bytes-allocated • mem-util-hash-hash-dmem-rec-allocation-count • mem-util-hash-hash-dmem-rec-free-count • mem-util-hash-hash-dmem-sideband-bytes-allocated • mem-util-hash-hash-dmem-sideband-allocation-count • mem-util-hash-hash-dmem-sideband-free-count • mem-util-encaps-ueid-bytes-allocated • mem-util-encaps-ueid-allocation-count • mem-util-encaps-ueid-free-count

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none">• mem-util-encaps-ueid-shared-bytes-allocated• mem-util-encaps-ueid-shared-bytes-allocated• mem-util-encaps-ueid-shared-allocation-count• mem-util-encaps-ueid-shared-free-count• mem-util-encaps-fabric-bytes-allocated• mem-util-encaps-fabric-allocation-count• mem-util-encaps-fabric-free-count• mem-util-services-nh-inline-jflow-sample-rr-(svcs)-bytes-allocated• mem-util-services-nh-inline-jflow-sample-rr-(svcs)-allocation-count• mem-util-services-nh-inline-jflow-sample-rr-(svcs)-free-count• mem-util-services-nh-inline-jflow-sample-nh-(svcs)-bytes-allocated• mem-util-services-nh-inline-jflow-sample-nh-(svcs)-allocation-count• mem-util-services-nh-inline-jflow-sample-nh-(svcs)-free-count

Table 27: gRPC Sensors (Continued)

Resource Path	Description
/junos/system/linecard/npu/memory/	<p>Sensor for network processing unit (NPU) memory, NPU memory utilization, and total memory available for each memory type.</p> <p>Supported on QFX10000 switches and PTX1000 routers starting with Junos OS Release 17.2R1.</p> <p>Supported on EX9200 switches starting with Junos OS Release 17.3R1.</p> <p>NOTE: Starting with Junos Release 17.4R1, FPC1 and FCP2 on PTX Series routers export data for NPU memory and NPU memory utilization. Previously, this sensor was supported only on FPC 3.</p> <p>Starting with Junos OS Release 18.3R1, EX4650 switches are supported.</p> <p>Starting with Junos OS Release 19.1R1, periodic streaming on PTX10002 routers is supported.</p> <p>Starting in Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers and PTX1000 and PTX10000 routers are supported on gRPC and gNMI services.</p> <p>The OpenConfig path is /components/component[name="FPC<fpc-id>:NPU<npu-id>"] /properties/property/</p> <p>You can also add the following to the end of the path to stream specific statistics for NPU memory:</p> <ul style="list-style-type: none"> • [name="mem-util-<memory-name>-size"]/value • [name="mem-util-<memory-name>-bytes-allocated"]/value • [name="mem-util-<memory-name>-utilization"]/value • [name="mem-util-<partition-name>-<app-name>-allocation-count"]/value • [name="mem-util-<partition-name>-<app-name>-bytes-allocated"]/value • [name="mem-util-<partition-name>-<app-name>-free-count"]/value <p>You can also add the following to the end of the path to stream specific statistics for NPU:</p>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • [name="util-<memory-name>-average-util"/>/value • [name="util-<memory-name>-highest-util"/>/value • [name="util-<memory-name>-lowest-util"/>/value • [name="util-<memory-name>-average-cache-hit-rate"/>/value • [name="util-<memory-name>-lowest-cache-hit-rate"/>/value • [name="util-<packet-identifier>-rate"/>/value <p>You can also export the following statistics for NPU memory for PTX routers only</p> <ul style="list-style-type: none"> • pfe_name • combined_pool_name • combined_size • combined_usage_cnt • combined_utilization • global_pool_name • global_usage_cnt • global_alloc_cnt • global_free_cnt • local_pool_name • local_usage_cnt • local_alloc_cnt • local_free_cnt

Table 27: gRPC Sensors (Continued)

Resource Path	Description
/junos/system/linecard/npu/memory/	<p>Sensor for NPU Memory utilization statistics.</p> <p>Shown below, statistics are exported for the default FPC (FPC0). Multiples FPCs are supported. The component values and property values are names (like interface names).</p> <p>Starting in Junos OS Evolved Release 19.4R1, streaming statistics using gRPC and gNMI services on PTX10008 routers is supported.</p> <p>Starting in Junos OS Release 20.2R1, INITIAL_SYNC statistics using gNMI services on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches are supported.</p> <p>Starting in Junos OS Evolved Release 21.4R1, streaming statistics by means of gRPC and gNMI is supported on PTX10001-36MR, PTX10004, and PTX10008 routers.</p> <p>Supported on PTX10003 routers starting with Junos OS Release 22.3R1.</p> <p>Statistics are exported in the following format: /components/component[name='FPC*:NPU*']/properties/property[name=<>]/state/value</p> <p>The list below shows the property names:</p> <ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-epp-mapid-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-epp-mapid-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-epp-mapid-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-l2domain-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-l2domain-allocated']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-l2domain-utilizationn']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-tunnell2domainhash00-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-tunnell2domainhash00-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-tunnell2domainhash00-utilization']/ • :/components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-tunnell2domainhash10-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-tunnell2domainhash10-allocatedd']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-tunnell2domainhash10-utilization']/ • :/components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-slu-my-mac-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-slu-my-mac-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-slu-my-mac-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-dlu-idb-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-dlu-idb-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-kht-dlu-idb-utilization']/ } • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-final-size']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-final-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-final-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-remap-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-remap-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-remap-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-refbits-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-refbits-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-refbits-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-nh-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-nh-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-nh-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-mpis-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-mpis-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-mpis-utilization']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-loadbal-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-loadbal-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-loadbal-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-egress-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-egress-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jnh-egress-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jtree-memory-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jtree-memory-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-jtree-memory-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-vfilter-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-vfilter-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-vfilter-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-phyfilter-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-phyfilter-allocated']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-phyfilter-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-action-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-action-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-action-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-tcam-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name=' mem-util-flt-tcam-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-tcam-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-0-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-0-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-0-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-1-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-1-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-1-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-2-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-2-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-2-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-3-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-3-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-3-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-4-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-4-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-fcv-blk-4-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-scv-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-scv-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-scv-utilization']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-0-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-0-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-0-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-1-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-1-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-1-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-2-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-2-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-2-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-3-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-3-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-3-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-4-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-4-allocated']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-4-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-5-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-5-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-5-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-6-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-6-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-6-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-7-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-7-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-0-bank-7-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-0-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-0-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-0-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-1-size']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-1-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-1-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-2-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-2-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-2-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-3-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-3-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-3-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-4-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-4-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-4-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-5-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-5-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-5-utilization']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-6-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-6-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-6-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-7-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-7-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-beta-1-bank-7-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-kht-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-kht-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-kht-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-bft-0-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-bft-0-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-bft-0-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-plt-size']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-plt-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-0-plt-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-kht-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-kht-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-kht-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-bft-0-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-bft-0-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-bft-0-utilization']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-plt-size']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-plt-allocated']/ • /components-memory/component[name='FPC0:NPU17']/properties/property[name='mem-util-flt-alpha-1-plt-utilization']/

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/junos/system/linecard/npu/utilization	<p>Sensor for NPU utilization on the Packet Forwarding Engine.</p> <p>Packet Forwarding Engine utilization is exported as a percentage using input notifications.</p> <p>The following packet statistics are also exported as part of this field:</p> <ul style="list-style-type: none"> • Loopback (pps) • Recirculation (pps) • WAN and host inject (pps) • ASIC to host (pps) <p>Shown below, statistics are exported for the default FPC (FPC0). Multiples FPCs are supported. The component values and property values are names (like interface names).</p> <p>Starting in Junos OS Evolved Release 19.4R1, streaming statistics using gRPC and gNMI services on PTX10008 routers is supported.</p> <p>The following statistics are exported:</p> <ul style="list-style-type: none"> • /components-utilization/component[name='FPC0:NPU17'] • /components-utilization/component[name='FPC0:NPU17']/properties/property[name='util-metric'] • /components-utilization/component[name='FPC0:NPU17']/properties/property[name='util-Loopback-packet-rate'] • components-utilization/component[name='FPC0:NPU17']/properties/property[name='util-Recirculation-packet-rate'] • /components-utilization/component[name='FPC0:NPU17']/properties/property[name='util-Wan and Host inject-packet-rate'] • /components-utilization/component[name='FPC0:NPU17']/properties/property[name='util-ASIC to host-packet-rate']

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/system/linecard/npu/utilization/	<p>Packet Forwarding Engine sensor for NPU processor utilization.</p> <p>Supported on MX Series routers with MPC10E-10C-MRATE and MPC10E-15C-MRATE line cards starting with Junos OS Release 19.3R1 for streaming telemetry information using gRPC services. This feature provides a different level of exported statistics in comparison to previous releases because it uses the OpenConfig AFT model.</p> <p>Supported on MX2010 and MX2020 routers with MX2K-MPC11E line cards starting with Junos OS Release 20.1R1 for streaming telemetry information using gRPC services.</p> <p>Supported on MX304 routers using Juniper proprietary gRPC and gNMI starting with Junos OS Release 22.1R1.</p> <p>Supported on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches starting with Junos OS Release 20.2R1 and later for INITIAL_SYNC statistics using gNMI services.</p> <p>You can also include the following to the end of the resource path for NPU utilization:</p> <ul style="list-style-type: none"> • util-metric • util-Disp 0 Pkts-packet-rate • util-Disp 0 Pkts-average-instructions-per-packet • util-Disp 0 Pkts-average-wait-cycles-per-packet • util-Disp 0 Pkts-average-cycles-per-packet • util-Disp 1 Pkts-packet-rate • util-Disp 1 Pkts-average-instructions-per-packet • util-Disp 1 Pkts-average-wait-cycles-per-packet • util-Disp 1 Pkts-average-cycles-per-packet • util-Disp 2 Pkts-packet-rate

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<ul style="list-style-type: none"> • util-Disp 2 Pkts-average-instructions-per-packet • util-Disp 2 Pkts-average-wait-cycles-per-packet • util-Disp 2 Pkts-average-cycles-per-packet • util-Disp 3 Pkts-packet-rate • util-Disp 3 Pkts-average-instructions-per-packet • util-Disp 3 Pkts-average-wait-cycles-per-packet • util-Disp 3 Pkts-average-cycles-per-packet • mem-util-EDMEM-average-util • mem-util-EDMEM-highest-util • mem-util-EDMEM-lowest-util • mem-util-EDMEM-average-cache-hit-rate • mem-util-EDMEM-highest-cache-hit-rate • mem-util-EDMEM-lowest-cache-hit-rate • mem-util-IDMEM-average-util • mem-util-IDMEM-highest-util • mem-util-IDMEM-lowest-util • mem-util-IDMEM-average-cache-hit-rate • mem-util-IDMEM-highest-cache-hit-rate • mem-util-IDMEM-lowest-cache-hit-rate • mem-util-Bulk DMEM-average-util • mem-util-Bulk DMEM-highest-util • mem-util-Bulk DMEM-lowest-util

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
	<ul style="list-style-type: none"> • mem-util-Bulk DMem-average-cache-hit-rate • mem-util-Bulk DMem-highest-cache-hit-rate • mem-util-Bulk DMem-lowest-cache-hit-rate
/junos/system/linecard/npu/utilization/	<p>Packet Forwarding Engine sensor for NPU processor utilization.</p> <p>Periodic streaming is supported on PTX10002 routers starting with Junos OS Release 19.1R1.</p> <p>Starting with Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers and PTX1000 and PTX10000 routers are supported on gRPC and gNMI services.</p> <p>Starting in Junos OS Evolved Release 21.4R1, streaming statistics by means of gRPC and gNMI is supported on PTX10001-36MR, PTX10004, and PTX10008 routers.</p>
/junos/system/linecard/optical	<p>Sensor for optical alarms. Configure this sensor for <i>et-type-fpc/pic/port</i> (100-Gigabit Ethernet) interfaces.</p> <p>Supported on ACX6360 Universal Metro, MX Series, and PTX Series routers with a CFP2-DCO optics module starting with Junos OS Release 18.3R1. This module provides a high-density, long-haul OTN transport solution with MACSec capability.</p> <p>Supported on MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches starting with Junos OS Release 19.2R1 on gRPC and gNMI services.</p>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
<code>/junos/system/linecard/otn</code>	<p>Sensor for G.709 optical transport network (OTN) alarms. Configure this sensor on <code>ot- type-fpc/pic/port</code> interfaces.</p> <p>Supported on ACX6360 Universal Metro, MX Series, and PTX Series routers with a CFP2-DCO optics module starting with Junos OS Release 18.3R1. This module provides a high-density, long-haul OTN transport solution with MACSec capability.</p> <p>Supported on MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches starting with Junos OS Release 19.2R1 on gRPC and gNMI services.</p>
<code>/junos/system/linecard/packet/usage/</code>	<p>Sensor for Packet Forwarding Engine error and drop statistics. Use these statistics to optimize traffic engineering and improve your network design.</p> <p>When you include the resource path <code>/junos/system/linecard/packet/usage/</code> in a subscription, statistics are streamed in the format:</p> <pre>/components/component[name='FPC0:NPU3']/properties/property[name='hwds-dlu-not-routable']/state/value</pre> <p>Supported on PTX1000 and PTX5000 routers and QFX10002-60C switches using Juniper proprietary gRPC starting with Junos OS Release 22.1R1.</p> <p>Supported on PTX10003 routers starting with Junos OS Evolved Release 22.3R1.</p>
<code>/junos/system/linecard/page-drops/page-drop/</code>	<p>Sensor for CoS support. Use this sensor to stream CoS page-drop counters and interface details from a device to a collector. Page drop statistics include page drop counter, interface name and queue details.</p> <p>Supported on PTX5000 routers starting with Junos OS Release 22.2R1.</p> <p>Supported on PTX10003 routers starting with Junos OS Evolved Release 22.3R1.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/system/linecard/qmon-sw/	<p>Sensor for congestion and latency monitoring statistics.</p> <p>Supported on QFX5100, QFX5110, and QFX5200 switches starting with Junos OS Release 18.2R1.</p> <p>Supported on QFX5120-48Y and EX4650 switches starting with Junos OS Release 18.3R1.</p> <p>Supported on EX4600 switches starting with Junos OS Release 18.4R1.</p> <p>Supported on MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches starting with Junos OS Release 19.2R1 on gRPC and gNMI services.</p> <p>Supported on QFX5200 switches starting with Junos OS Release 19.2R1 for streaming telemetry information using gNMI services.</p> <p>Periodic streaming using gRPC services with EX4300-MP switches is supported starting with Junos OS Release 19.4R1.</p> <p>Periodic streaming using gRPC services is supported on EX3400 switches starting with Junos OS Release 20.2R1.</p> <p>INITIAL_SYNC statistics using gNMI services is supported on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches starting with Junos OS Release 20.2R1.</p> <p>Periodic streaming is supported on QFX5120-48YM switches starting with Junos OS Release 20.4R1.</p> <p>Periodic streaming is supported on PTX5000, PTX1000, PTX10002, PTX10008, and PTX10016 routers and QFX10002, QFX10008, and QFX10016 switches starting with Junos OS Release 21.2R1.</p>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
<code>/junos/system/linecard/services/inline-jflow</code>	<p>Sensor for inline active flow monitoring services statistics.</p> <p>Supported on MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches starting with Junos OS Release 19.2R1 on gRPC and gNMI services.</p> <p>When configuring inline active flow monitoring in Junos, you can apply version 9 or IPFIX flow templates to define a flow record template suitable for IPv4 or IPv6 MPLS and bridging traffic. For more information, see https://www.juniper.net/documentation/us/en/software/junos/flow-monitoring/topics/concept/services-configuring-flow-aggregation-to-use-version-9-flow-templates.html.</p> <p>Supported on MX Series operating with MPC10E-15C-MRATE line-rate cards starting with Junos OS Release 19.2R1.</p> <p>Supported on MX240, MX480, and MX960 routers starting with Junos OS Release 19.3R1 for exporting telemetry information using gNMI services. This feature includes support to export telemetry data for integration with AFTTelemetry and LibTelemetry libraries with the OpenConfig model openconfig-aft.</p> <p>Periodic streaming using gNMI services on MX2K-MPC11E line cards on MX2010 and MX2020 routers is supported starting with Junos OS Release 20.1R1.</p> <p>Periodic streaming using gRPC services on PTX10008 routers is supported starting with Junos OS Evolved Release 20.1R1.</p>
<code>/network-instances/network-instance[instance-name='name']/protocols/protocol/evpn/irb-interfaces/</code>	<p>Local integrated routing and bridging (IRB) interface information sensor.</p> <p>Use the https://apps.juniper.net/ydm-explorer/ tool to see leafs for this resource path.</p> <p>Starting with Junos OS Release 20.2R1, streaming statistics is supported using gRPC services with QFX5100, QFX5110, QFX5120, QFX5200, QFX10002-60C, QFX10002, QFX10008, and QFX10016 switches.</p>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/network-instances/network-instance[instance-name='name']/protocols/protocol/evpn/vxlan-tunnel-end-point/	<p>Overlay VX-LAN tunnel information sensor. This sensor also delivers VTEP information ON_CHANGE leafs:</p> <ul style="list-style-type: none"> • source_ip_address • remote_ip_address • status • mode • nexthop-index • event-type • source-interface <p>Starting with Junos OS Release 20.2R1, streaming statistics is supported using gRPC services with QFX5100, QFX5110, QFX5120, QFX5200, QFX10002-60C, QFX10002, QFX10008, and QFX10016 switches.</p>
/network-instances/network-instance[instance-name='name']/mac_db/entries/entry/	<p>EVPN MAC table information sensor.</p> <p>Use the Telemetry Explorer tool to see leafs for this resource path.</p> <p>Starting with Junos OS Release 20.2R1, streaming statistics is supported using gRPC services with QFX5100, QFX5110, QFX5120, QFX5200, QFX10002-60C, QFX10002, QFX10008, and QFX10016 switches.</p>
/network-instances/network-instance[instance-name='name']/macip_db/entries/entry/	<p>MAC-IP or ARP-ND table sensor.</p> <p>Use the Telemetry Explorer tool to see leafs for this resource path.</p> <p>Starting with Junos OS Release 20.2R1, streaming statistics is supported using gRPC services with QFX5100, QFX5110, QFX5120, QFX5200, QFX10002-60C, QFX10002, QFX10008, and QFX10016 switches.</p>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
/junos/system/linecard/optics/	<p>Sensor for various optical interface performance metrics, such as transmit and receive power levels.</p> <p>The following leaves streamed with the /junos/system/linecard/optics/ resource path return a value of -Inf dB milliwatt (dBm) when the power is 0 milliwatt (mW)), To view these statistics from the Junos CLI, use the operational mode command show interface diagnostics optics.</p> <ul style="list-style-type: none"> • /interfaces/interface/optics/ laser_output_power_high_alarm_threshold_dbm • /interfaces/interface/optics/ laser_output_power_low_alarm_threshold_dbm • /interfaces/interface/optics/ laser_output_power_high_warning_threshold_dbm • /interfaces/interface/optics/ laser_output_power_low_warning_threshold_dbm • /interfaces/interface/optics/ laser_rx_power_high_alarm_threshold_dbm • /interfaces/interface/optics/ laser_rx_power_low_alarm_threshold_dbm • /interfaces/interface/optics/ laser_rx_power_high_warning_threshold_dbm • /interfaces/interface/optics/ laser_rx_power_low_warning_threshold_dbm • /interfaces/interface/optics/lanediags/lane/ lane_laser_output_power_dbm • /interfaces/interface/optics/lanediags/lane/ lane_laser_receiver_power_dbm <p>The following resource paths are also supported on MX10004 starting with Junos OS Release 22.3R1:</p>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /junos/system/linecard/optics/optics-diag[if-name =] • /junos/system/linecard/optics/optics-diag/if-name • /junos/system/linecard/optics/optics-diag/snmp-if-index <p>MX10004 routers support these endpoints:</p> <ul style="list-style-type: none"> • module_temp • module_temp_high_alarm_threshold • module_temp_low_alarm_threshold • module_temp_high_warning_threshold • module_temp_low_warning_threshold • laser_output_power_high_alarm_threshold_dbm • laser_output_power_low_alarm_threshold_dbm • laser_output_power_high_warning_threshold_dbm • laser_output_power_low_warning_threshold_dbm • laser_rx_power_high_alarm_threshold_dbm • laser_rx_power_low_alarm_threshold_dbm • laser_rx_power_high_warning_threshold_dbm • laser_rx_power_low_warning_threshold_dbm • laser_bias_current_high_alarm_threshold • laser_bias_current_low_alarm_threshold • laser_bias_current_high_warning_threshold • laser_bias_current_low_warning_threshold • module_temp_high_alarm

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
	<ul style="list-style-type: none"> • module_temp_low_alarm • module_temp_high_warning • module_temp_low_warning <p>Supported on QFX10000 switches starting with Junos OS Release 17.2R1.</p> <p>Supported on PTX1000 routers and EX9200 switches starting with Junos OS Release 17.3R1.</p> <p>Supported on EX4650 switches starting with Junos OS Release 18.3R1.</p> <p>Supported on MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches starting with Junos OS Release 19.2R1 on gRPC and gNMI services.</p> <p>Supported on MX10 routers, PTX1000 and PTX10000 routers, and QFX5100 and QFX5200 switches starting with Junos OS Release 19.2R1 on gRPC and gNMI services.</p> <p>Supported on MX10008 routers starting with Junos OS Release 22.1R1 using Juniper proprietary gRPC.</p> <p>Supported on MX10004 routers starting with Junos OS Release 22.3R1 using Juniper proprietary gRPC.</p> <p>Supported on ACX7100-32C, ACX7100-48L, and ACX7024 routers starting with Junos OS Evolved Release 22.3R1.</p>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
<pre>/mpls/lsp-constrained-path/tunnels/ tunnel[name='foo-name',source='foo-source']/p2p- tunnel-attributes/p2p-primary-paths[name='foo- path']/state/name</pre>	<p>Sensor to export the path name for ingress point-to-point LSPs, point-to-multipoint LSPs, bypass LSPs, and dynamically created LSPs.</p> <p>This sensor is supported on indicated platforms up to and including Junos OS Release 17.3R1. See the following resource paths for LSP support in Junos OS Release 17.4R1 and higher:</p> <ul style="list-style-type: none"> • /network-instances/network-instance[name='instance-name']/mpls/lsp-constrained-path/tunnels/tunnel/p2p-tunnel-attributes/p2p-primary-paths/ • /network-instances/network-instance[name='instance-name']/mpls/signaling-protocols/rsvp-te/sessions/session/state/notify-status <p>Supported on PTX Series routers, MX Series routers , and QFX10002, QFX10008, and QFX10016 switches starting with Junos OS Release 17.2R1.</p>
<pre>/mpls/lsp-constrained-path/tunnels/ tunnel[name='foo-name',source='foo-source']/p2p- tunnel-attributes/p2p-primary-paths[name='foo- path']/lsp-instances[index='local-index']/state/</pre>	<p>Sensor to export LSP properties for ingress point-to-point LSPs, point-to-multipoint LSPs, bypass LSPs, and dynamically created LSPs</p> <p>Supported on PTX Series routers, MX Series routers, and QFX10002, QFX10008, and QFX10016 switches starting with Junos OS Release 17.2R1.</p> <p>The following end paths are also supported for the resource path:</p> <ul style="list-style-type: none"> • bandwidth • metric • max-average-bandwidth • explicit-route-objects • record-route-objects
<pre>/mpls/signaling-protocols/ldp/lsp-transit-policies/ lsp-transit-policy/state/counters</pre>	<p>Sensor to export statistics for LDP LSP transit traffic.</p> <p>Supported on MX Series routers starting with Junos OS Release 20.2R1.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/mpls/signaling-protocols/ldp/lsp-ingress-policies/ lsp-ingress-policy/state/counters	Sensor to export statistics for LDP LSP ingress traffic. Supported on MX Series routers starting with Junos OS Release 20.2R1.
/mpls/signaling-protocols/ldp/p2mp-lsps/p2mp-lsp/ state/counters	Sensor to export statistics for multipoint LDP LSP traffic. Supported on MX Series routers starting with Junos OS Release 20.2R1.
/mpls/signalling-protocols/ldp/p2mp-interfaces/ p2mp-interface/state/counters/	Sensor to export statistics for multipoint LDP egress traffic per interface. Supported on MX Series routers starting with Junos OS Release 20.2R1.
/mpls/signalling-protocols/ldp/p2mp-egress- interfaces/p2mp-interface/state/counters/	Sensor to export statistics for multipoint LDP egress traffic per interface. Supported only on MPC10E-10C-MRATE, MPC10E-15C-MRATE, and MX2K-MPC11E line cards line cards on MX Series routers starting with Junos OS Release 20.3R1.
/mpls/signalling-protocols/ldp/p2mp-interfaces/ p2mp-interface/	Sensor to export statistics for multipoint LDP ingress traffic per interface. Supported on MX Series routers starting with Junos OS Release 20.2R1.

Table 27: gRPC Sensors (Continued)

Resource Path	Description
<code>/mpls/lsp/signaling-protocols/rsvp-te/sessions/session[local-index='foo-index']/state/notify-status</code>	<p>Sensor to export statistics for ingress point-to-point LSPs, point-to-multipoint LSPs, bypass LSPs, and dynamically created LSPs.</p> <p>ON_CHANGE support for LSP events is only activated when the reporting interval is set to 0 in the subscription request.</p> <p>Supported on PTX Series routers, MX Series routers, and QFX10002, QFX10008, and QFX10016 switches starting with Junos OS Release 17.2R1.</p> <p>The following events are exported under this resource path:</p> <ul style="list-style-type: none">• PATHERR_RECEIVED<ul style="list-style-type: none">• TTL_EXPIRED• NON_RSVP_CAPABLE_ROUTER• RESVTEAR_RECEIVED• PATH_MTU_CHANGE

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/network-instances/network-instance/mpls/signaling-protocols/rsvp-te/	<p>Sensor to export events for ingress point-to-point LSPs, point-to-multipoint LSPs, bypass LSPs, and dynamically created LSPs.</p> <p>Starting in Junos OS Evolved Release 19.2R1, PTX10003 routers support streaming statistics.</p> <p>The following end paths are also supported:</p> <ul style="list-style-type: none"> • interface-attributes/interface/bandwidth-reservations/state/active-reservations-count • interface-attributes/interface/bandwidth-reservations/state/available-bandwidth • interface-attributes/interface/bandwidth-reservations/state/highwater-mark • interface-attributes/interface/bandwidth-reservations/state/reserved-bandwidth • interface-attributes/interface/counters/in-ack-messages • interface-attributes/interface/counters/in-hello-messages • interface-attributes/interface/counters/in-path-messages • interface-attributes/interface/counters/in-path-tear-messages • interface-attributes/interface/counters/in-reservation-error-messages • interface-attributes/interface/counters/in-reservation-messages • interface-attributes/interface/counters/in-reservation-tear-messages • interface-attributes/interface/counters/in-srefresh-messages • interface-attributes/interface/counters/out-path-tear-messages • interface-attributes/interface/counters/out-ack-messages • interface-attributes/interface/counters/out-hello-messages

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<ul style="list-style-type: none">• interface-attributes/interface/counters/out-path-messages• interface-attributes/interface/counters/out-reservation-error-messages• interface-attributes/interface/counters/out-reservation-messages• interface-attributes/interface/counters/out-reservation-tear-messages• interface-attributes/interface/counters/out-srefresh-messages• neighbors/neighbor/state/neighbor-status• sessions/session/record-route-objects/record-route-object• sessions/session/state/destination-address• sessions/session/state/label-in• sessions/session/state/label-out• sessions/session/state/lsp-id

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/mpls/signaling-protocols/segment-routing/	<p>Sensor for traffic statistics for both ingress IP traffic and transit MPLS traffic..</p> <p>Supported on MX Series and PTX Series routers starting with Junos OS Release 18.3R1.</p> <p>The following end points are also supported and specify BGP Segment Routing traffic Engineering (SR-TE) transit statistics:</p> <ul style="list-style-type: none"> • /sr-te-bsid-policies/sr-te-bsid-policy[binding-sid='80001', to-address='foo-to' color='foo-color']/state/counters[name='oc-xxx']/packets • /sr-te-bsid-policies/sr-te-bsid-policy[binding-sid='80001', to-address='foo-to' color='foo-color']/state/counters[name='oc-xxx']/bytes <p>The following end points are also supported and specify BGP Segment Routing traffic Engineering (SR-TE) ingress statistics:</p> <ul style="list-style-type: none"> • /sr-te-ip-policies/sr-te-ip-policy[to-address='foo-to' color='foo-color']/state/counters[name='oc-xxx']/packets • /sr-te-ip-policies/sr-te-ip-policy[to-address='foo-to' color='foo-color']/state/counters[name='oc-xxx']/bytes <p>In addition to configuring the sensor, you must enable statistics collection using the statistics statement at the <code>[[edit protocols</code> https://www.juniper.net/documentation/us/en/software/junos/cli-reference/topics/ref/statement/source-packet-routing-edit-protocols.html <code>telemetry statistics]</code> hierarchy level.</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
/junos/system/linecard/packet/usage/	<p>Sensor for Packet Forwarding Engine Statistics. This sensor exports statistics for counters and provides visibility into Packet Forwarding Engine error and drop statistics.</p> <p>This sensor is supported starting on MX Series and PTX Series routers starting with Junos OS Release 17.4R1.</p> <p>Starting in Junos OS Evolved Release 19.1R1, PTX10003 routers are supported.</p> <p>Starting in Junos OS Release 19.2R1, MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5200 switches are supported on gRPC and gNMI services.</p> <p>Starting with Junos OS Evolved Release 19.4R1, periodic streaming using gNMI services with PTX10003 routers is supported.</p>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
/junos/system/linecard/packet/usage/	<p>Sensor for Packet Forwarding Engine Statistics. This sensor exports statistics and provides visibility into Packet Forwarding Engine error and drop statistics. Statistics include counters (CC, CPU, and NPU) for traffic data. Note that NPU statistics are different than those streamed from the sensors <code>/junos/system/linecard/npu/memory/</code> and <code>/junos/system/linecard/npu/utilization/</code>. Sensor output is comparable to the output using the operational mode command show pfe statistics traffic.</p> <p>Shown below, statistics are exported for the default FPC (FPC0). Multiples FPCs are supported. The component values and property values are names (like interface names).</p> <p>Starting in Junos OS Evolved Release 19.4R1, streaming statistics using gRPC and gNMI services on PTX10008 routers is supported.</p> <p>Starting in Junos OS Release 20.2R1, INITIAL_SYNC statistics using gNMI services on MX960, MX2008, MX2010, MX2020, PTX1000, PTX5000 routers, PTX10000 line of routers, and QFX5100 and QFX5200 switches are supported.</p> <p>Starting in Junos OS Evolved Release 21.4R1, streaming statistics by means of gRPC and gNMI is supported on PTX10001-36MR, PTX10004, and PTX10008 routers.</p> <p>The following paths are also supported:</p> <ul style="list-style-type: none"> • <code>:/components/component[name='FPC0:CC0']/properties/property[name='ts-input-packets']/</code> • <code>/components/component[name='FPC0:CC0']/properties/property[name='ts-output-packets']/</code> • <code>//components/component[name='FPC0:CC0']/properties/property[name='ts-input-packets-pps']/</code> • <code>/components/component[name='FPC0:CC0']/properties/property[name='ts-output-packets-pps']/</code> • <code>/components/component[name='FPC0:CC0']/properties/property[name='ts-fabric-input-packets']/</code>

Table 27: gRPC Sensors (Continued)

Resource Path	Description
	<ul style="list-style-type: none"> • /components/component[name='FPC0:CC0']/properties/property[name='ts-fabric-input-packets-pps']/ • /components/component[name='FPC0:CC0']/properties/property[name='ts-fabric-output-packets']/ • /components/component[name='FPC0:CC0']/properties/property[name='ts-fabric-output-packets-pps']/ • /components/component[name='FPC0:CPU0']/properties/property[name='lts-input-packets']/ • /components/component[name='FPC0:CPU0']/properties/property[name='lts-output-packets']/ • /components/component[name='FPC0:CPU0']/properties/property[name='lts-sw-input-control-drops']/ • /components/component[name='FPC0:CPU0']/properties/property[name='lts-sw-input-high-drops']/ • /components/component[name='FPC0:CPU0']/properties/property[name='lts-sw-input-medium-drops']/ • /components/component[name='FPC0:CPU0']/properties/property[name='lts-sw-input-low-drops']/ • /components/component[name='FPC0:CPU0']/properties/property[name='lts-sw-output-low-drops']/ • /components/component[name='FPC0:CPU0']/properties/property[name='lts-hw-input-drops']/ • /components/component[name='FPC0:NPU0']/properties/property[name='hwdsNormal']/ • /components/component[name='FPC0:NPU0']/properties/property[name='hwds-data-error']/ • /components/component[name='FPC0:NPU0']/properties/property[name='hwds-tcp-error']/

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	<ul style="list-style-type: none">• /components/component[name='FPC0:NPU0']/properties/property[name='hwds-illegal-nh']/• /components/component[name='FPC0:NPU0']/properties/property[name='hwds-invalid-iif']/ //components/component[name='FPC0:NPU0']/properties/property[name='hwds-fabric']/

Table 27: gRPC Sensors (Continued)

Resource Path	Description
<code>/junos/system/linecard/packet-capture</code>	<p>Sensor for secure packet capture.</p> <p>Starting in Junos OS Release 21.2R1 on EX4400 switches, we support secure packet capture. You can use this feature to capture packets from a device and send them over a secure channel to an external collector (in the cloud) for monitoring and analysis. The maximum size of the packet you can capture is 128 bytes, including the packet header and the data within. Network professionals use real-time packet capture data to troubleshoot complex issues such as network and performance degradation and poor end-user experience.</p> <p>To use secure packet capture, include the <code>/junos/system/linecard/packet-capture</code> resource path using a Junos RPC call.</p> <p>For ingress packet capture, include the <code>packet-capture</code> option in the existing firewall filter configuration at the</p> <pre>[edit firewall family family-name filter filter-name term match-term then packet-capture]</pre> <p>hierarchy level. Do this before you send packet capture sensor data to the collector and remove the <code>packet-capture</code> configuration after data is sent to the collector. After the capture is done, ingress packets with the filter match conditions are trapped to the CPU. The trapped packets then go to the collector over a secure channel in JTI-specified format in key-value pairs by means of Remote Procedure Call (gRPC) transport.</p> <p>For egress packet capture on physical interfaces (ge-*, xe-*, mge-*, and et-*), include "packet-capture-telemetry," "egress," and "interface <interface-name>" at the <code>[edit forwarding-options]</code> hierarchy level. For example:</p> <pre>set forwarding-options packet-capture-telemetry egress interface ge-0/0/0</pre> <pre>set forwarding-options packet-capture-telemetry egress interface ge-0/0/10</pre> <p>You can add multiple interfaces on the device for egress packet capture. When configured, host-bound egress packets are captured</p>

Table 27: gRPC Sensors *(Continued)*

Resource Path	Description
	from the interface and sent to the collector. As with the ingress configuration, remove the configuration when packet capture is not required.

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
/interfaces/	<p>Sensor for device monitoring.</p> <p>To stream statistics, use the resource path /interfaces/ in a subscription to export statistics to a collector in the following format: <code>/interfaces/interface[name='et-*/*/*']/</code>.</p> <p>Exported statistics include the following:</p> <ul style="list-style-type: none"> • <code>/interfaces/interface[name='et-*/*/*']/init_time</code> • <code>/interfaces/interface[name='et-*/*/*']/oper-status</code> • <code>/interfaces/interface[name='et-*/*/*']/parent_ae_name</code> • <code>/interfaces/interface[name='et-*/*/*']/in-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/in-octets</code> • <code>/interfaces/interface[name='et-*/*/*']/in-unicast-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/in-multicast-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/in-broadcast-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/in-pause-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/out-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/out-octets</code> • <code>/interfaces/interface[name='et-*/*/*']/out-unicast-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/out-multicast-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/out-broadcast-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/out-pause-pkts</code> • <code>/interfaces/interface[name='et-*/*/*']/in-errors</code> • <code>/interfaces/interface[name='et-*/*/*']/in-discards</code> • <code>/interfaces/interface[name='et-*/*/*']/out-errors</code>

Table 27: gRPC Sensors (*Continued*)

Resource Path	Description
	<ul style="list-style-type: none"> • <code>/interfaces/interface[name='et-*/*/*']/out-discards</code> <p>Supported starting in Junos OS 22.3R1 on PTX10003 routers.</p>
<code>/qos/interfaces/interface/output/queues/queue/state/</code>	<p>Sensor for CoS telemetry support.</p> <p>To stream statistics, use the resource path <code>/qos/interfaces/interface/output/queues/queue/state/</code> in a subscription to retrieve statistics from a router to a collector in the following format: <code>/qos/interfaces/interface[interface-id='xe-1/1/5:0']</code> .</p> <p>The following end points are supported:</p> <ul style="list-style-type: none"> • <code>/queues/queue[name='0']/state/transmit-pkts</code> • <code>/queues/queue[name='0']/state/transmit-octets</code> • <code>/queues/queue[name='0']/state/dropped-pkts</code> <p>Starting in Junos OS Evolved Release 21.4R1, streaming statistics by means of gRPC and gNMI is supported on PTX10001-36MR, PTX10003, PTX10004, PTX10008, and PTX10016 routers.</p>
<code>/system/alarms/alarm</code>	<p>INITIAL_SYNC support for OpenConfig data model <code>openconfig-platform.yang</code> and <code>openconfig-alarms.yang</code>. This feature lets the collector have a complete view of the current state of every sensor it is subscribed to. INITIAL_SYNC requires that at least one copy of all the sensors be sent to the collector.</p> <p>Starting in Junos OS Evolved Release 21.4R1, streaming statistics by means of gRPC and gNMI is supported on PTX10001-36MR, PTX10003, PTX10004, PTX10008, and PTX10016 routers.</p>

Table 28: Broadband Edge gRPC Sensors

resource path	Description
<p>/junos/system/subscriber-management/chassis/virtual-chassis-ports/virtual-chassis-port</p> <p>/junos/system/subscriber-management/chassis/virtual-chassis-ports/virtual-chassis-port[vcp-interface-name=<i>vcp-interface-port-string</i>] (to specify the interface name)</p>	<p>Virtual chassis port counter sensor.</p> <p>The sensor includes these statistics:</p> <ul style="list-style-type: none"> • Input packets • Output packets • Input bytes • Output bytes <p>Starting with Junos OS Release 20.2R1, streaming statistics from a virtual chassis is supported using gRPC services with MX480, MX960, MX10003, MX2010, and MX2020 routers.</p>
<p>/junos/system/subscriber_management/dynamic-interfaces/interface-sets/meta-data/interface[sid-id=<i>'sid-value'</i>]/</p>	<p>Sensor for subscriber interface information.</p> <p>ON-CHANGE streaming is supported.</p> <p>The following end paths are supported:</p> <ul style="list-style-type: none"> • interface-index-The system assigned interface index for the interface. • session-type-The type of client session (e.g VLAN, DHCP, PPPoE). • user-name-The login name for this interface and session. • profile-name-The name of the client profile used to create the interface. • underlying-interface-name-The name of the associated underlying interface. • cvlan-tag-The innermost VLAN tag value associated with the interface. • svlan-tag-The outermost VLAN tag value associated with the interface.

Table 28: Broadband Edge gRPC Sensors (Continued)

resource path	Description
<pre>/junos/system/subscriber_management/dynamic- interfaces/interface-sets/meta-data/ interface[sid-id='sid-value']/</pre>	<p>Sensor for actual accounting statistics for dynamic subscriber interfaces.</p> <p>The following end paths are supported:</p> <ul style="list-style-type: none"> • ip-in-packets-The number of actual transit IPv4 & IPv6 packets received by the interface. • ip-out-packets-The number of actual transit IPv4 & IPv6 packets sent to the interface. • ip-in-bytes-The number of actual transit IPv4 & IPv6 bytes received by the interface. • ip-out-bytes-The number of actual transit IPv4 & IPv6 bytes received by the interface. • ipv6-in-packets-The number of actual transit IPv6 packets received by the interface. • ipv6-out-packets-The number of actual transit IPv6 packets sent to the interface. • ipv6-in-bytes-The number of actual transit IPv6 bytes received by the interface. • ipv6-out-bytes-The number of actual transit IPv6 bytes sent to the interface.
<pre>/junos/system/linecard/ddos/</pre>	<p>This PFE sensor exports the statistics of DDOS from MPC1, MPC2, MPC3, MPC5, MPC6, MPC7, MPC8, and MPC9 line cards.</p>

Change History Table

Feature support is determined by the platform and release you are using. Use [Feature Explorer](#) to determine if a feature is supported on your platform.

Release	Description
20.1R1	Starting with Junos Release 20.1R1, gNMI service for streaming telemetry sensors for Packet Forwarding Engine statistics is supported on MX2K-MPC11E line cards on MX2010 and MX2020 routers.

19.4R1	Starting with Junos OS Release 19.4R1, gRPC service for streaming Packet Forwarding Engine and Routing Engine statistics is supported on EX4300-MP switches.
19.3R1	Starting with Junos OS Release 19.3R1, gNMI services for streaming Packet Forwarding Engine statistics is supported on MX240, MX480 and MX960 routers.
19.3R1	Starting with Junos OS Release 19.3R1, gRPC service for exporting statistics is supported on MX Series routers hosting MPC10E-10C-MRATE and MPC10E-15C-MRATE line cards. The resource paths <code>/junos/system/linecard/cpu/memory/</code> , <code>/junos/system/linecard/npu/memory/</code> , and <code>/junos/system/linecard/npu/utilization/</code> can be updated to call out individual sensors (leaves) and their respective paths for better clarity.
19.3R1	Starting with Junos OS Evolved Release 19.3R1, gRPC service for exporting statistics is supported on QFX5220-128C and QFX5220-32CD switches.
19.2R1	Starting with Junos OS Release 19.2R1, SRX4100, SRX4200, SRX4600, SRX5400, SRX5600, SRX5800, and vSRX Virtual Firewall Series Services Gateways are supported.
19.2R1	Starting with Junos OS Release 19.2R1, gNMI services for streaming Packet Forwarding Engine statistics is supported on MX960, MX2008, MX2010 and MX2020 routers, PTX1000 and PTX10000 routers, and QFX5100 and QFX5200 switches.
19.2R1	Starting with Junos OS Release 19.2R1, gNMI services for streaming statistics is supported on QFX5110, QFX5120, QFX5200 and QFX5210 switches.
19.2R1	Starting with Junos OS Release 19.3R1, gNMI services for streaming and ON_CHANGE export of Routing Engine statistics is supported on MX960, MX2010, MX2020, PTX5000, PTX1000, and PTX10000 routers.
19.1R1 EVO	Starting in Junos OS Evolved Release 19.1R1, OpenConfig (OC) and Junos Telemetry Interface (JTI) are supported. Both gRPC APIs and the customer-facing CLI remain the same as for the Junos OS. As was standard for Junos OS, Network Agent (NA) and OC packages are part of the Junos OS Evolved image.
19.1R1	Starting with Junos OS Evolved 19.1R1, Packet Forwarding Engine sensors on PTX10003 routers are also supported.
18.4R1	Starting with Junos OS Release 18.4R1, MX480, MX960, MX2010, MX2020, MX2008 and MX-ELM routers are also supported.

18.4R1	Starting with Junos OS Release 18.4R1, BGP operational states are aligned and compliant with OpenConfig data model openconfig-bgp-operational.yang . To stream BGP operational states, use the resource path /network-instances/network-instance/protocols/protocol/bgp/ . Previously, the path was /bgp/ .
18.3R1	Starting with Junos OS Release 18.3R1, ON_CHANGE streaming of LLDP telemetry sensor information is supported through gRPC for MX Series and PTX Series routers.
18.3R1	Starting with Junos OS Release 18.3R1, QFX5120-48Y and EX4650 switches are also supported.
18.3R1	Starting with Junos OS Release 18.4R1, EX4600 switches are also supported.
18.2R1	Starting with Junos OS Release 18.2R1, PTX10002 routers are also supported.
18.1R1	Starting with Junos OS Release 18.1R1, QFX5210-64C switches and QFX5100 switches are also supported.
18.1R1	Starting with Junos OS Release 18.1R1, ON_CHANGE streaming of ARP, ND, and IP sensor information associated with interfaces is supported through gRPC for MX Series routers and PTX Series routers.
17.4R1	Starting with Junos OS Release 17.4R1, PTX10016 routers and virtual MX Series (vMX) routers are also supported.
17.3R1	Starting with Junos OS Release 17.3R1, QFX5110 switches, EX4600, EX4600-VC, and EX9200 switches and the Routing and Control Board (RCB) on PTX3000 routers are also supported.
17.3R1	Starting with Junos OS Release 17.3R1, broadband edge (BBE) gRPC sensors are supported.
17.3R1	In Junos OS Release 17.3R1, broadband edge (BBE) gRPC sensor <code>/junos/system/subscriber-management/client-protocols/dhcp/v4/routing-instances/routing-instance[ri-name=' <i>routing-instance-name</i>'] /server/statistics/</code> the only value supported for <i>routing-instance-name</i> is default.
17.3R1	In Junos OS Release 17.3R1, broadband edge (BBE) gRPC sensor <code>/junos/system/subscriber-management/client-ancpinstance[ri-name=' <i>routing-instance-name</i>'] /server/statistics/</code> the only value supported for <i>routing-instance-name</i> is default.
17.3R1	In Junos OS Release 17.3R1, broadband edge (BBE) gRPC sensor <code>/junos/system/subscriber-management/client-protocols/dhcp/v4/routing-instances/routing-instance[ri-name=' <i>routing-instance-name</i>']/relay/statistics/</code> the only value supported for the value <i>routing-instance-name</i> is default.

17.3R1	In Junos OS Release 17.3R1, broadband edge (BBE) gRPC sensor /junos/system/subscriber-management/client-protocols/dhcp/v6/ routing-instances/routing-instance[ri-name= ' <i>routing-instance-name</i> ']/server/statistics the only value supported for <i>routing-instance-name</i> is default.
17.3R1	In Junos OS Release 17.3R1, broadband edge (BBE) gRPC sensor /junos/system/subscriber-management/client-protocols/dhcp/v6/ routing-instances/routing-instance[ri-name= ' <i>routing-instance-name</i> ']/relay/statistics the only value supported for <i>routing-instance-name</i> is default.
17.2R1	Starting with JunosOS Release 17.2R1, QFX10002, QFX10008, and QFX10016 switches, QFX5200 switches, and PTX1000 and PTX10008 routers are also supported.
16.1R3	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 on MX Series routers and PTX3000 and PTX5000 routers.

J-Insight Device Monitor Overview

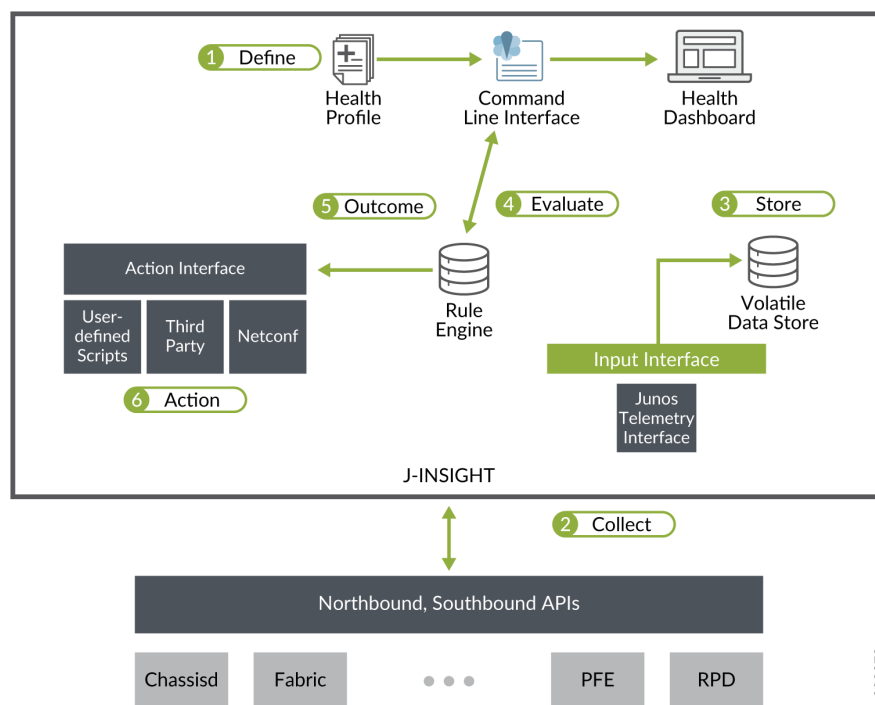
IN THIS SECTION

- [Understanding How J-Insight Health Monitoring Works | 336](#)
- [Understanding How J-Insight Fault Monitoring Works | 337](#)

As networks become increasingly complex, the need to adopt features that simplify the process of monitoring, maintaining, and improving the overall health of your networking devices becomes increasingly critical to delivering services in a more predictable and manageable way.

J-Insight is a data-driven device monitoring solution that provides visibility and insight into the health of a running system. Starting with Junos OS Release 18.2R1, the J-Insight framework facilitates real-time monitoring of system resources for FPC FRUs. It also has been integrated with the existing connectivity error management infrastructure to normalize error detection, monitoring, and reporting. The long-term goal for the architectural design of the J-Insight device monitor is depicted in [Figure 10 on page 336](#).

Figure 10: Long-term High-level Architecture for J-Insight



J-Insight is an on-premise system application that uses the Junos Telemetry Interface to continuously collect data that is reflective of the current state and health of the device component being monitored.

Understanding How J-Insight Health Monitoring Works

As part of this initial release, the J-Insight health monitor supports the following process flow (see [Figure 10 on page 336](#)):

1. Consumes a pre-defined static health profile. The health profile is not user-configurable through the Junos OS CLI.
2. Using the Junos Telemetry Interface (JTI) framework, subscribes to health KPIs specified in the default health profile. J-Insight health monitor subscribes to JTI sensors using a standard interface. Health monitor subscription and reporting is disabled, by default, and can be enabled through the Junos OS CLI. Starting with Junos OS Release 18.2R1, the following health KPIs are supported for MX-based FPCs:
 - CPU utilization
 - Temperature sensors

- PFE memory utilization
 - Fabric reachability
3. Collates the JTI data streams collected from various sub-systems.
 4. Evaluates the health data against configured thresholds and reports the health status.

Understanding How J-Insight Fault Monitoring Works

Starting with Junos OS Release 18.2R1, J-Insight utilizes the connectivity error management infrastructure to normalize error detection, monitoring, and reporting. Through this infrastructure, J-Insight also provides the capability to define data-driven fault policies. Each module can define error properties by reading a DST/capability file. The fault monitoring capability is available by default in Junos OS and cannot be enabled or disabled through the CLI.

Each error is defined by the following properties:

- **URI**—Error identifier. Each error is uniquely identified with an error ID that is represented as a Uniform Resource Identifier (URI).
- **Error**—Error name.
- **Scope**—Error scope. An error scope provides a level of classification above the error category. Examples of error scope values include: pfe and board.
- **Category**—Error category. An error category categories errors into various subgroups under a specific error scope level. Examples of error category values include: memory, processing, and storage.
- **Details**—Description for the error.
- **Count**—The number of times error instances have occurred.
- **Clear count**—The number of times error instances have been cleared.
- **Support**—Support details for the error type.

RELATED DOCUMENTATION

[J-Insight Device Monitor Basic Configuration](#) | 338

J-Insight Device Monitor Basic Configuration

IN THIS SECTION

- [Before you Begin | 338](#)
- [J-Insight Health Monitoring | 341](#)
- [J-Insight Fault Monitoring | 341](#)

Before you Begin



NOTE: If you're running Junos OS Evolved software, you do not need to perform the procedures in this "Before you Begin" section.

J-Insight requires that your Junos OS device supports the Junos Telemetry Interface (JTI). For information about JTI, see the [Junos Telemetry Interface User Guide](#). To use J-Insight, you must first complete the following steps:

1. Use the `show agent sensors` command to verify whether or not J-Insight has successfully subscribed to sensors on which it is dependent.

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

    Name                : sensor_1000
    Resource              : /junos/events/event[id='CHASSISD_SNMP_TRAP7']/
    Version              : 1.0
    Sensor-id            : 539528115
    Subscription-ID       : 1000
    Parent-Sensor-Name    : Not applicable
    Component(s)         : eventd

Profile Information :
```

```

Name                : export_1000
Reporting-interval   : 0
Payload-size         : 5000
Format               : GPB

```

Sensor Information :

```

Name                : sensor_1001
Resource             : /junos/system/cmerror/configuration/
Version              : 1.0
Sensor-id            : 539528114
Subscription-ID       : 1001
Parent-Sensor-Name    : Not applicable
Component(s)         : PFE

```

Profile Information :

```

Name                : export_1001
Reporting-interval   : 6
Payload-size         : 5000
Format               : GPB

```

Sensor Information :

```

Name                : sensor_1002
Resource             : /junos/system/cmerror/counters/
Version              : 1.0
Sensor-id            : 539528113
Subscription-ID       : 1002
Parent-Sensor-Name    : Not applicable
Component(s)         : PFE

```

Profile Information :

```

Name                : export_1002
Reporting-interval   : 6
Payload-size         : 5000
Format               : GPB

```

Sensor Information :

```

Name                : sensor_1003

```

```

Resource           : /components/
Version            : 1.0
Sensor-id          : 539528112
Subscription-ID     : 1003
Parent-Sensor-Name : Not applicable
Component(s)       : chassisd

```

Profile Information :

```

Name               : export_1003
Reporting-interval : 6
Payload-size       : 5000
Format             : GPB

```

Sensor Information :

```

Name               : sensor_1004
Resource           : /junos/services/health-monitor/config/
Version            : 1.0
Sensor-id          : 539528119
Subscription-ID     : 1004
Parent-Sensor-Name : Not applicable
Component(s)       : PFE

```

Profile Information :

```

Name               : export_1004
Reporting-interval : 7
Payload-size       : 5000
Format             : GPB

```

Sensor Information :

```

Name               : sensor_1005
Resource           : /junos/services/health-monitor/data/
Version            : 1.0
Sensor-id          : 539528118
Subscription-ID     : 1005
Parent-Sensor-Name : Not applicable
Component(s)       : PFE

```

Profile Information :

Name	: export_1005
Reporting-interval	: 7
Payload-size	: 5000
Format	: GPB

J-Insight Health Monitoring

- To enable the J-Insight health monitor:

```
user@host# set services jinsightd subscribe health-monitor
```

- To disable the J-Insight health monitor:

```
user@host# delete services jinsightd subscribe health-monitor
```

- To display the J-Insight health monitor results:

```
user@host> show system health-monitor [fpc fpc-slot slot-number]
```

J-Insight Fault Monitoring

IN THIS SECTION

- [Chassis-level Configuration Commands | 342](#)
- [Trace Commands | 342](#)
- [Clear & Show Commands | 342](#)

Starting with Junos OS Evolved Release 19.1R1, J-Insight fault monitoring support is added for CB, chassis, fan, FPC, FPM, PDU, PIC, PSM, RE and SIB FRUs.

Chassis-level Configuration Commands

The Junos OS resiliency feature provides debugging capabilities in the case of device component failure. You can configure Packet Forwarding Engine (PFE)-related error levels on FRUs such as FPCs. Using the *error* configuration statement, you can set an automatic recovery action for each severity and configure the actions to perform when a specified threshold is reached.

For more information, see the [Chassis-Level User Guide](#).

Trace Commands

- (Junos OS only) To enable J-Insight trace options for debugging:

```
user@host# set services jinsightd traceoptions flag trace-option
```

- (Junos OS Evolved only) You can view collected J-Insight traces with the **show trace application jinsightd** command, and remove inactive J-Insight tracing sessions with the **clear trace application jinsightd** command.

Clear & Show Commands

- To clear all system errors or a specific error denoted by the error ID Uniform Resource Identifier (URI) for a specific FPC:

```
user@host> clear chassis fpc errors fpc-slot slot-number [ all | error-id error-id- uri]
```

- To display information on alarms that have been triggered by faults:

```
user@host> show chassis alarms
```

- To display summary or detailed information about the active errors based on FRU, error scope, or error category:

```
user@host> show system errors active [[fru slot-number] | [detail [fru slot-number [scope error-scope ] [category error-category ]]]]
```

- To display a summary of the number of detected errors and recovery actions taken based on severity level:

```
user@host> show system errors count
```

- To display information about a detected error based on its error ID URI:

```
user@host> show system errors error-id error-id-uri
```

- To display detailed information about the detected errors based on the FRU:

```
user@host> show system errors fru detail [fru slot-number]
```

RELATED DOCUMENTATION

| [J-Insight Device Monitor Overview](#) | 335