Telemetry Configuration Guide for Cisco 8000 Series Routers, IOS XR Release 25.1.x, 25.2.x

First Published: 2025-03-31

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

CHAPTER 1

New and Changed Feature Information

This section lists all the new and changed features for the *Telemetry Configuration Guide for Cisco 8000 Series Routers*.

Telemetry Features Added or Modified in IOS XR Release 25.x.x

Table 1: New and Changed Telemetry Features

Feature	Description	Changed in Release	Where Documented
Stream telemetry data for interface statistics	No new features introduced	Release 25.2.1	Stream telemetry for IPv4 and IPv6 data on network interfaces, on page 64

CHAPTER 2

YANG Data Models for Telemetry Features

This chapter provides information about the YANG data models for Telemetry features.

Using YANG Data Models

Cisco IOS XR supports a programmatic way of configuring and collecting operational data of a network device using YANG data models. Although configurations using CLIs are easier and human-readable, automating the configuration using model-driven programmability results in scalability.

The data models are available in the release image, and are also published in the Github repository. Navigate to the release folder of interest to view the list of supported data models and their definitions. Each data model defines a complete and cohesive model, or augments an existing data model with additional XPaths. To view a comprehensive list of the data models supported in a release, navigate to the **Available-Content.md** file in the repository.

You can also view the data model definitions using the YANG Data Models Navigator tool. This GUI-based and easy-to-use tool helps you explore the nuances of the data model and view the dependencies between various containers in the model. You can view the list of models supported across Cisco IOS XR releases and platforms, locate a specific model, view the containers and their respective lists, leaves, and leaf lists presented visually in a tree structure. This visual tree form helps you get insights into nodes that can help you automate your network.

To get started with using the data models, see the *Programmability Configuration Guide*.

CHAPTER 3

Scale-Up Your Network Monitoring Strategy Using Telemetry

Are you monitoring your network using traditional polling methods such as SNMP, Syslog, and CLI? If yes, does the data that you extract from your network help you answer these questions?

- What percentage of the network bandwidth does the network traffic currently consume?
- Do all the links in the network run at a hundred percent utilization rate?
- If an unmanned router fails, is the network operator notified in real time about the issue and its related consequences?
- Is the CPU over- or under-utilized?
- Can the efficiency of the network be calculated based on traffic and data loss?
- What are the possible performance issues that cause traffic loss or network latency?
- How do you proactively prevent issues that may arise? Does the data support the study of network patterns in real time?

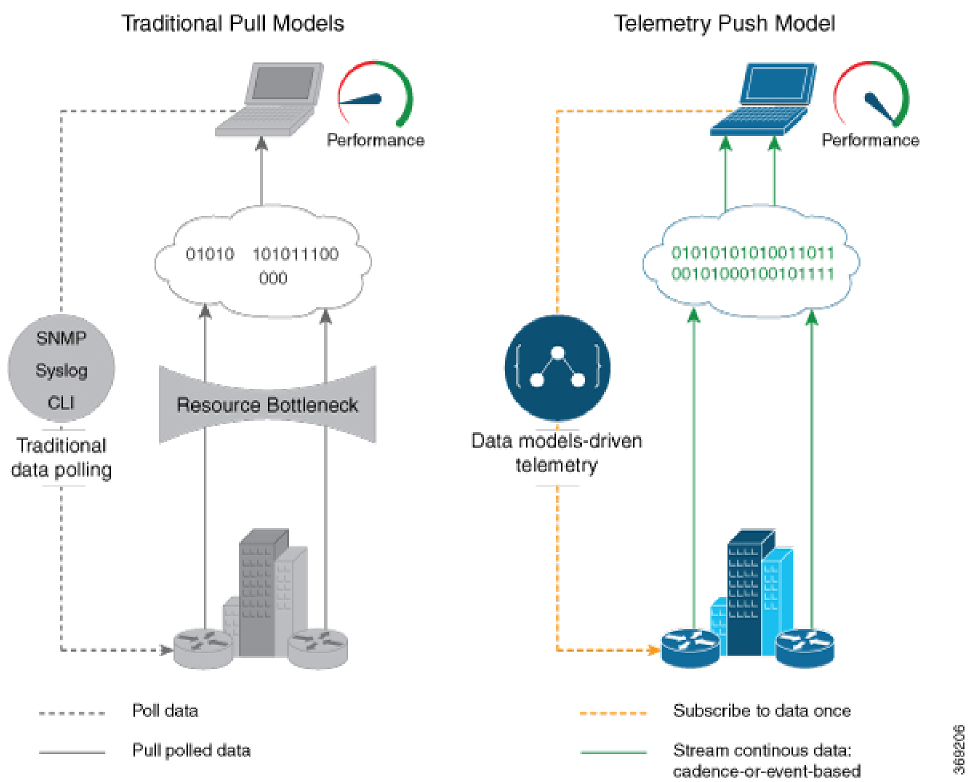
These traditional methods use a *pull* model to request information at regular intervals. The data that you collect may help you to efficiently monitor your network of a manageable size. However, as your network grows in complexity and scale, the data that you poll may be insufficient for efficient and effective monitoring. Additionally, the polling methods are resource-intensive, and network operators face information gaps in the data that they collect. With the pull model, the network device (the server) sends data only when the data collector (the client) requests it. Initiating such requests requires continual manual intervention. This manual intervention makes this model unsuitable, and limits automation and the ability to scale. It inhibits the visibility of the network and therefore provides inefficient control of the network. You need monitoring strategy that adds resiliency and stability to your network.

Telemetry does just that. Telemetry uses a *push* model that automatically streams data from a network device. Instead of a collector requesting data at periodic intervals, the network device streams operational data in real time.

Telemetry focuses on the power of scale, speed, and automation. With the power of flexibility, you can select data of interest from the routers and transmit it in a structured format to remote management stations for monitoring. Using the finer granularity and higher frequency of data available through telemetry, DevOps (development and operations) engineers in your organization can quickly locate and investigate issues as soon as they occur. They can, thus, collaborate to monitor and have better control over the network.

The following image shows the comparative benefits of streaming telemetry data using the telemetry push model over traditional pull models. The pull models create resource bottlenecks that prevent retrieving valuable operational data from the router. On the other hand, the push model is designed to remove such bottlenecks and deliver data efficiently.

Figure 1: Comparison Between Traditional Pull Models and Telemetry Push Model



Watch this video to see how telemetry data can unlock the intelligence of data in your network to proactively predict and troubleshoot issues.

Note Starting from Cisco IOS XR, Release 7.0.1, Telemetry is part of the base image (<platform>-mini-x.iso). In earlier releases, Telemetry was part of the Manageability package (<platform>-mgbl-3.0.0.0-<release>.x86_64.rpm).

This article describes the benefits of using telemetry data and the various methods to stream meaningful data from your network device:

Benefits of Shifting Network Monitoring from Pull Models to Telemetry Push Model

Real-time telemetry data is useful in:

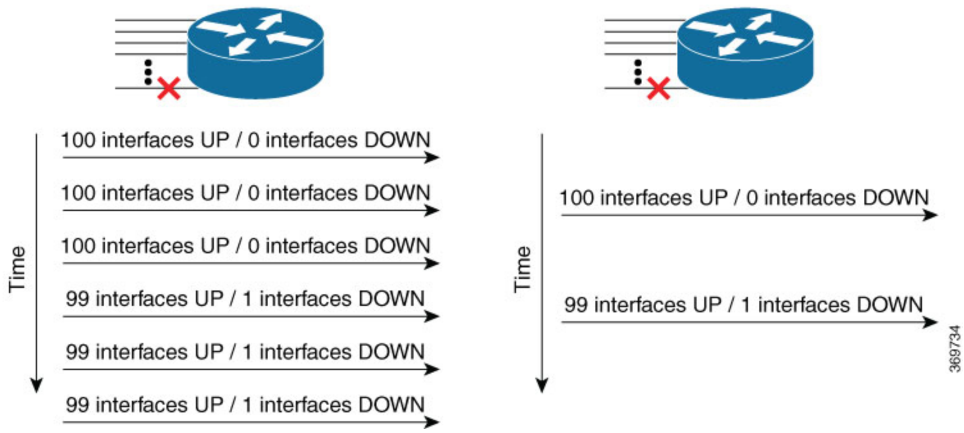
- **Managing network remotely:** The primary benefit of telemetry is the ability it offers you as an end user to monitor the state of a network element remotely. After the network is deployed, you cannot be physically present at the network site to find out what works, and what is cumbersome. With telemetry, those insights can be analyzed, leveraged, and acted upon from a remote location.
- **Optimizing traffic:** When link utilization and packet drops in a network are monitored at frequent intervals, it is easier to add or remove links, re-direct traffic, modify policing, and so on. With technologies like fast reroute, the network can switch to a new path and re-route faster than the traditional SNMP poll interval mechanism. Streaming telemetry data helps in providing quick response time for faster transport of traffic.
- **Preventive troubleshooting:** Network state indicators, network statistics, and critical infrastructure information are exposed to the application layer, where they are used to enhance operational performance and to reduce troubleshooting time. The finer granularity and higher frequency of data available through telemetry enables better performance monitoring and therefore, better troubleshooting.
- **Visualizing data:** Telemetry data acts as a data lake that analytics toolchains and applications use to visualize valuable insights into your network deployments.
- **Monitoring and controlling distributed devices:** The monitoring function is decoupled from the storage and analysis functions. This decoupling helps to reduce device dependency, while providing flexibility to transform data using pipelines. These pipelines are utilities that consume telemetry data, transform it, and forward the resulting content to a downstream, typically off-the-shelf, consumer. The supported downstream consumers include Apache Kafka, Influxdata, Prometheus, and Grafana.

Streaming telemetry, thus, converts the monitoring process into a Big Data proposition that enables the rapid extraction and analysis of massive data sets to improve decision-making.

Review Mechanisms to Stream Telemetry Data from a Router to a Destination

Telemetry data can be streamed using either cadence-driven or event-driven mechanisms.

Figure 2: Cadence-driven and Event-driven Telemetry

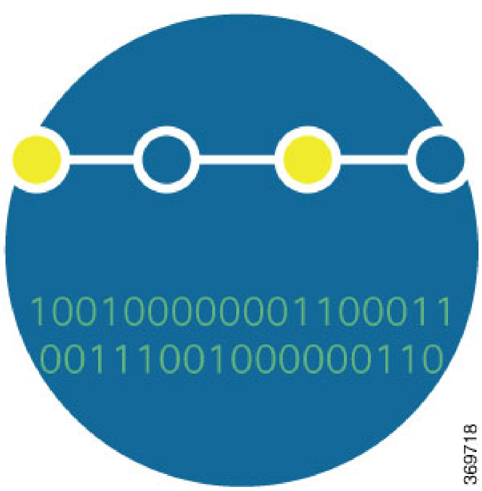


Cadence-driven Telemetry

Cadence-driven telemetry continually streams data (operational statistics and state transitions) at a configured cadence. The higher frequency of the data that is continuously streamed helps you closely identify emerging patterns in the network.

The following image shows a continuous stream of data after a configured time interval:

Figure 3: Cadence-driven Telemetry

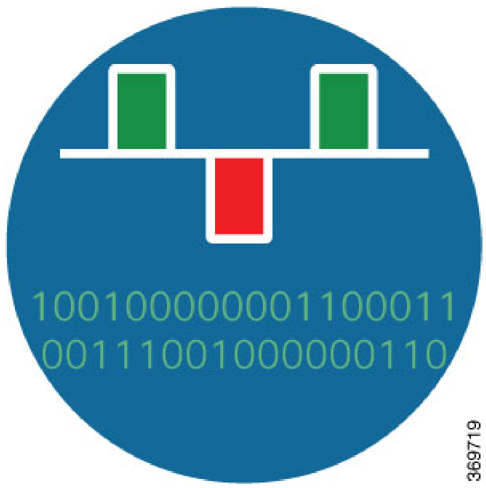


Event-driven Telemetry

Event-driven telemetry optimizes data that is collected at the receiver and streams data only when a state transition occurs and thus optimizes data that is collected at the receiver. For example, EDT streams data about interface state transitions, IP route updates, and so on.

The following image shows a stream of data after a state change:

Figure 4: Event-driven Telemetry



Learn About the Elements that Enable Streaming Telemetry Data

These elements are the building blocks in enabling telemetry in a network.

Sensor Path

Table 2: Feature History Table

Feature Name	Release Information	Description
Stream Digital Optical Monitoring (DOM) Data	Release 7.3.1	This feature streams fiber optic transceiver parameters such as optical input or output levels, temperature, laser bias current, supply voltage, receiver power, bias threshold in real-time. This helps network operators to easily locate a fiber link failure, thereby simplifying the maintenance process, and improving overall system reliability. Sensor paths introduced for this feature are: Cisco-IOS-XR-dwdm-ui-oper:dwdm/ports/port/info/optics-info Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info

The sensor path describes a YANG path or a subset of data definitions in a YANG data model within a container. In a YANG model, the sensor path can be specified to end at any level in the container hierarchy.

A YANG module defines a data model through the data of the router, and the hierarchical organization and constraints on that data.

YANG defines four node types. Each node has a name. Depending on the node type, the node either defines a value or contains a set of child nodes. The nodes types for data modeling are:

- leaf node - contains a single value of a specific type
- leaf-list node - contains a sequence of leaf nodes
- list node - contains a sequence of leaf-list entries, each of which is uniquely identified by one or more key leaves
- container node - contains a grouping of related nodes that have only child nodes, which can be any of the four node types

To get started with using the data models, see the *Programmability Configuration Guide*.

The following table shows few examples of sensor paths:

Table 3: Sensor Paths

Feature	Sensor Path
CPU	`Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization`
Memory	`Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary`

Interface	`Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters`
	`Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/data-rate`
	`openconfig-interfaces:interfaces/interface`

Optical power levels	`Cisco-IOS-XR-dwdm-ui-oper:dwdm/ports/port/info/optics-info` `Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info`
Node summary	`Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary`
Forwarding information base (FIB)	`Cisco-IOS-XR-fib-common-oper:fib-statistics/nodes/node/drops` `Cisco-IOS-XR-fib-common-oper:fib/nodes/node/protocols/protocol/vrfs/vrf/summary`

MPLS Traffic engineering (MPLS-TE)	`Cisco-IOS-XR-mpls-te-oper:mpls-te/tunnels/summary`
	`Cisco-IOS-XR-ip-rsvp-oper:rsvp/interface-briefs/interface-brief`
	`Cisco-IOS-XR-mpls-te-oper:mpls-te/fast-reroute/protections/protection`
	`Cisco-IOS-XR-mpls-te-oper:mpls-te/signalling-counters/signalling-summary`
	`Cisco-IOS-XR-mpls-te-oper:mpls-te/p2p-p2mp-tunnel/tunnel-heads/tunnel-head`

MPLS Label distribution protocol (MPLS-LDP)	`Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/nodes/node/bindings-summary-all`
	`Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/global/active/default-vrf/summary`
	`Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/nodes/node/default-vrf/neighbors/neighbor`

Feature	Sensor Path
Routing	Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/statistics-global Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/neighbors/neighbor Cisco-IOS-XR-ip-rib-ipv4-oper:rib/rib-table-ids/rib-table-id/summary-protos/summary-proto Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/levels/level/adjacencies/adjacency Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance/instance-active/default-vrf/process-info Cisco-IOS-XR-ip-rib-ipv6-oper:ipv6-rib/rib-table-ids/rib-table-id/summary-protos/summary-proto

Note Use specific paths to avoid streaming data that you may not be interested. For example, if you want to stream information about only the summary of MPLS-TE, use `sensor-path Cisco-IOS-XR-mpls-te-oper:mpls-te/autotunnel/mesh/summary` instead of `sensor-path Cisco-IOS-XR-mpls-te-oper:mpls-te` sensor path.

The router streams telemetry data at predefined gather points in the data model even if sensor-path configuration is to an individual leaf. The gather points are collection units; collection always happens at that level for operational data.

Starting from release 7.2.1, the router supports the following sensor-path resolutions:

- Streaming data at the leaf-level or at the container-level under a gather point for cadence-based subscriptions.

  If a subscritpion has multiple sensor-paths that resolve to the same gather point and have the same cadence and encoding, data is pushed in a single collection stream for all the leaves. For example:

```
telemetry model-driven
 sensor-group intf-stats
  sensor-path
Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/interface-statistics/full-interface-stats/bytes-sent

  sensor-path
Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface/interface-statistics/full-interface-stats/bytes-received

 !
 subscription intf-stats
  sensor-group-id intf-stats sample-interval 10000
 !
!
end
```

  This subscription pushes one message with two leaves because the gather point `full-interface-stats` is same for both the sensor-paths `bytes-sent` and `bytes-received`. This grouping of the leaves happens at the subscription level. If these paths are configured under different subscriptions, data is streamed as different collections with separate messages each including one leaf `bytes-sent` or `bytes-received`.

- For event-driven subscriptions, streaming is always at the gather point in the model, even if specific leaves or leaf is configured as sensor-path. There is configuration to restrict streaming specific leaves for event-driven subscriptions. If this configuration is used, the sensor-path of the configured leaf streams data even if there is a change in one of its adjacent leaves. This indicates that even if there is no change in value of the configured leaf, data can stream out to the collector. The collector must be set to check if the leaf value changed before taking action on the streamed data.

```
telemetry model-driven
include select-leaves-on-events
```

Note It is not recommended to configure sensor-paths with the same gather point into different subscriptions.

In the sensor path configuration, the schema node identifier can be configured with or without a leading slash.

An MDT-capable device, such as a router, associates the sensor path to the nearest container path in the model. The router encodes and streams the container path within a single telemetry message. A receiver receives data about all the containers and leaf nodes at and below this container path. The router streams telemetry data, for one or more sensor-paths, at the configured frequency (Cadence-driven Telemetry, on page 8), or when an event occurs (Event-driven Telemetry, on page 8), to one or more collectors through subscribed sessions.

Subscription

A subscription binds one or more sensor paths and destinations.

The collector uses the subscription to receive updates about the state of data on the router. A subscription can consist of one or more sensor paths. The data for the paths that you have subscribed starts streaming until the session is terminated by the collector or the telemetry subscription configuration is removed to cancel the subscription.

The following example shows subscription `SUB1` that associates a sensor-group, sample interval and destination group.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription SUB1
Router(config-model-driven-subs)#sensor-group-id SGROUP1 sample-interval 10000
Router(config-model-driven-subs)#strict-timer
```

Note With a `strict-timer` configured for the sample interval, the data collection starts exactly at the configured time interval allowing a more deterministic behavior to stream data. In 32-bit platforms, `strict-timer` can be configured only under the subscription. Whereas, 64-bit platforms support configurtion at global level in addition to the subscription level. However, configuring at the global level will affect all configured subscriptions.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#strict-timer
```

Encoder

Data that is streamed from a router can be encoded using one of these formats:

- **GPB encoding:** Configuring for GPB encoding requires metadata in the form of compiled `.proto` files. A `.proto` file describes the GPB message format which is used to stream data. The `.proto` files are available at Cisco Network Telemetry Proto in Github.
  - **Compact GPB encoding:** Data is streamed in a compressed format and not in a self-descriptive format. A `.proto` file corresponding to each sensor-path must be used by the collector to decode the streamed data.
  - **Self-describing GPB encoding:** Data streamed for each sensor path is in a self-describing and ASCII text format. A single `.proto` file, `telemetry.proto`, is used by the collector to decode any sensor path data. Self-describing GPB encoding is easier to manage because it needs single `.proto` file to decode any sensor path data, even though the message size is large.
- **JSON encoding:** Data is streamed in strings of keys and its values in a human-readable format.

Transport

In the telemetry push model, the router streams telemetry data using a transport protocol. The generated data is encapsulated into the desired format using encoders.

Model-Driven Telemetry (MDT) data is streamed through these supported transport protocols:

- Google Protocol RPC (gRPC): used for both dial-in and dial-out modes.

Note gRPC protocol is not supported over Multiprotocol Label Switching (MPLS) including `explicit-null` label.

- Transmission Control Protocol (TCP): used for only dial-out mode.
- User Datagram Protocol (UDP): used for only dial-out mode. Because UDP is connectionless, the UDP destination is shown as `Active` irrespective of the state of the collector.

  UDP for Telemetry is not recommended for production networks as it doesn't support models that send messages larger than the UDP size limit of 65507 bytes.

  If a message is dropped by the network before it reaches the collector, the protocol does not resend the data.

Note Telemetry data is streamed out of the router using an Extensible Manageability Services Deamon (emsd) process. The data of interest is subscribed through subscriptions and streamed through gRPC, TCP or UDP sessions. However, a combination of gRPC, TCP and UDP sessions with more than 150 active sessions leads to emsd crash or process restart.

gRPC Network Management Interface

gRPC Network Management Interface (gNMI) is a gRPC-based network management protocol used to modify, install or delete configuration from network devices. It is also used to view operational data, control and generate telemetry streams from a target device to a data collection system. It uses a single protocol to manage configurations and stream telemetry data from network devices.

For the list of gNMI RPCs, see the *Programmability Configuration Guide*.

gNMI Subscription Modes

gNMI defines 3 modes for a streaming subscription that indicates how the router must return data in a subscription:

- A `SAMPLE` mode is cadence-based subscription supported for all the operational models.
- An `ON_CHANGE` mode is event-based subscription. In this mode, only the state leaf supports on_change events.
- A `TARGET_DEFINED` mode allows the target to determine the best type of subscription to be created on a per-leaf basis.

When a client creates a subscription specifying the `TARGET_DEFINED` mode, the target, here the router, determine the best type of subscription to be created on a per-leaf basis. If the path specified within the message refers to some leaves which are event-driven, then an `ON_CHANGE` subscription is created.

Note In Cisco IOS XR Release 7.2.1, the `TARGET_DEFINED` subscription mode is supported only for sensor paths of OpenConfig model; native model is not supported. The supported models are: OC Interfaces, OC Telemetry, OC Shell Util, OC System NTP and OC Platform.

An initial synchronization is established with all the leaves. If a new client has the same request information, then the initial synchronization is sent to all the clients connected at that point. This indicates that if multiple clients request the same subscription information, then the initial synchronization is resent even to the older connections.

The **telemetry model-driven gnmi-target-defined** command can be used to determine the cadence for the leaves (set to be cadence-driven) using the following prameters:

- **cadence-factor:** Multiplier factor for cadence of target-defined subscriptions. The range is 1 to 10. The default value is 2.
- **minimum-cadence:** Minimum cadence for target-defined subscriptions in seconds. The range is 1 to 65535. The default value is 30 seconds.

If cadence is specified as part of the gNMI request, this cadence is used for the first collection of data. Once the collection time is calculated, use the following formula to calculate the cadence.

```
Cadence = Maximum (Global minimum-cadence defined, (Collection time for one collection * cadence-factor))
```

If cadence is not specified as part of the request, the default value of 30 secs is used. This value can be modified using the following commands:

```
telemetry model-driven
gnmi-target-defined minimum-cadence 90
gnmi-target-defined cadence-factor 6
!
!
```

For more information about gNMI, see Github.

Note At the start of the collection, target-defined subscriptions adjust its own sample cadence interval (initially set as 30 seconds) based on the first few iterations of collection. The sample interval is adjusted to `first collection time * target-defined cadence-factor`.

gRPC Network Operations Interface

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based microservices for executing operational commands on network devices. Extensible Manageability Services (EMS) gNOI is the Cisco IOS XR implementation of gNOI. gNOI uses gRPC as the transport protocol and the configuration is same as that of gRPC.

For the list of gNOI RPCs, see the *Programmability Configuration Guide*.

TLS Authentication

The gRPC protocol supports Transport Layer Security (TLS) for encrypting data. By default, model-driven telemetry uses TLS to dial-out.

When TLS is enabled, the server sends a certificate to authenticate it with the collector. The collector validates the certificate verifying which certificate authority has signed it and generates session keys to encrypt the session.

To bypass the TLS option, use **grpc no-tls** command.

Note Although TLS provides secure communication between servers and clients, TLS version 1.0 may pose a security threat. You can now disable TLS version 1.0 using the **grpc tlsv1-disable** command.

Dial-out scenario

- Verify that the gRPC protocol is configured for dial-out mode, as it supports TLS by default.
- Copy the TLS certificate to the `/misc/config/grpc/dialout/` path on the router. This certificate is used to authenticate the server with the collector.
- In the certificate, set the `Common Name` (CN) to match the command, which defaults to the IP address of the destination if not specified.

```
protocol grpc tls-hostname
```

- Disable TLS 1.0: For security reasons, you can disable TLS version 1.0 using the **tlsv1-disable** command .

```
grpc
            tlsv1-disable
```

This output shows the certificate that gRPC uses to establish a dialout session:

```
Router#run
[node:]$ls -l /misc/config/grpc/dialout/
total 4
-rw-r--r-- 1 root root 4017 dialout.pem
```

Dial-in scenario

- Use the certificate generated by EMS located in `/misc/config/grpc/[ems.pem|ca.cert|ems.key]`. To use your own certificates, replace them with the same names in the grpc directory and restart the process.

Filter Telemetry Data Using Regex Key

Table 4: Feature History Table

Feature Name	Release Information	Description
Filter Telemetry Data Using Regex Keys in Sensor Paths	Release 7.4.1	Streaming huge telemetry data can create congestion in the network. With this feature, you can use the regular expression (regex) keys in the sensor path configuration on the router. The keys limit the amount of data that can be streamed, thereby ensuring better bandwidth utilization.

You can stream telemetry data from your network device using sensor paths and subscriptions.

The regular expression (regex) keys are used in sensor paths to limit the amount of data getting streamed from the router. The keys can be specified for any lists in the sensor paths that are subscribed. This allows you to filter data at the source (router) instead of filtering data at the collector. Regex keys help in better bandwidth utilization because only the data of interest is streamed from the router. Regex keys are supported for native models, open-config (OC) models and events.

Note Filtering data using regex key is not supported for System admin data models.

The syntax **re** in the sensor path indicates a filtered data using regex key. The characters '*', '.', '[', ']' and '\' are supported. For example, `re'Gig.*'` matches all Gigabit interfaces.

Note Telemetry supports only the POSIX regular expressions.

Example: Regex Key Filtering on Gigabit Interfaces

The sensor path with specific keys:

```
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
[interface-name='GigabitEthernet0/0/0/0']/latest/generic-counters
```

To this sensor path, apply the regex key to stream data to match GigabitEthernet0/0/0/0, GigabitEthernet0/0/0/1, GigabitEthernet0/0/0/2 interfaces:

```
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface
[interface-name=re'GigabitEthernet0/0/0/[0-2]']/latest/generic-counters
```

Example: Regex Key Filtering on IP addresses

The sensor path with specific keys:

```
openconfig-network-instance:network-instances/network-instance/afts/ipv4-unicast/
ipv4-entry[prefix='100.100.100.1/32']/state
```

To this sensor path, apply regex keys to matche all IP addresses that starts with `100.100.100`. In this example, the asterix (*) entry matches a range of addresses from 1 to 256.

```
openconfig-network-instance:network-instances/network-instance/afts/ipv4-unicast/
ipv4-entry[prefix=re'100\\.100\\.100.*']/state
```

Explore the Methods to Establish a Telemetry Session

A telemetry session can be initiated using: either the dial-out mode or the dial-in mode. Although the modes to establish a telemetry session are different, both modes use the same data model and stream the same data.

Dial-Out Mode

In a *dial-out* mode, the router dials out to the receiver to establish a subscription-based telemetry session. Because the router initiates the connection, there is no need to manage the ports for inbound traffic. In this default mode of operation, the protocols you use to establish a session gives you the flexibility to chose between simplicity (TCP) and security (gRPC). A simple protocol requires only accessibility to the socket on the collector. A secure protocol, additionally, offers security capabilities to authenticate and encrypt the session. You can, therefore, secure your collector, and establish a much advanced method of communication with the router. If the connection between the router and the destination is lost, the router re-establishes the connection with the destination and continues to push data again. However, data transmitted during the time of reconnection is lost.

To explore the dial-out mode, and to create a dial-out session, see Establish a Model-Driven Telemetry Session from a Router to a Collector, on page 19.

Dial-In Mode

In a *dial-in* mode, a collector dials in to the router, and subscribes dynamically to one or more sensor paths specified in a subscription. The router is open for connections from the collector. This mode is useful to establish a single channel of communication with the router. Because the collector establishes the session, there is no need to create destinations in the configuration. Additionally, the protocol (gRPC) used to establish a session provides advanced security capabilities to authenticate and encrypt the session. If the connection between the router and the collector is lost, the session is cancelled. The collector must reconnect to the router to restart streaming data. Only gRPC supports dial-in session.

To explore the dial-in mode, and to create a dial-in session, see Establish a Model-Driven Telemetry Session from a Collector to a Router, on page 29.

Identify the Telemetry Session Suitable for Your Network

The transport protocols and encoding formats in your network help you determine which mode is suitable for your needs. The encoding efficiency is determined by the space that data occupies on the wire, memory utilization, and the amount of data that you plan to stream from the router.

- Use TCP dial-out mode if you plan to stream telemetry data using a simple setup with a single router and collector. It is simple to configure and does not require extensive knowledge about protocols. It removes the need to manage ports for inbound connections.
- Use gRPC dial-out mode if your setup involves scaling out to many devices or needs encryption of your data. This mode removes the need to manage ports for inbound connections.
- Use gRPC dial-in mode if you are already using gRPCin your network and you want your sessions to be dynamic without having the data streamed to fixed destinations. This mode is convenient if you prefer a centralized way configuring your network and requesting operational data.

CHAPTER 4

Establish a Model-Driven Telemetry Session from a Router to a Collector

Streaming telemetry is a new paradigm in monitoring the health of a network. It provides a mechanism to efficiently stream configuration and operational data of interest from Cisco IOS XR routers. This streamed data is transmitted in a structured format to remote management stations for monitoring and troubleshooting purposes.

With telemetry data, you create a data lake. Analyzing this data, you proactively monitor your network, monitor utilization of CPU and memory, identify patterns, troubleshoot your network in a predictive manner, and devise strategies to create a resilient network using automation.

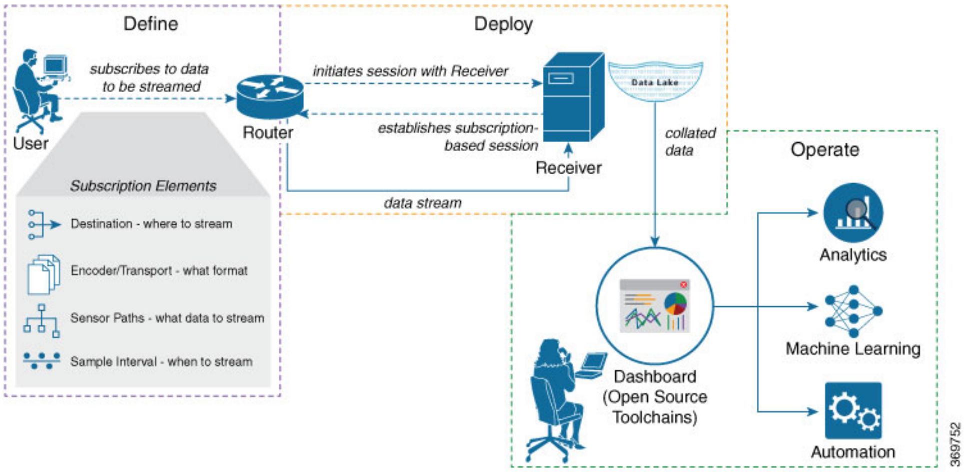
Telemetry works on a subscription model where you subscribe to the data of interest in the form of sensor paths. The sensor paths describes OpenConfig data models or native Cisco data models. You can access the OpenConfig and Native data models for telemetry from Github, a software development platform that provides hosting services for version control. You choose who initiates the subscription by establishing a telemetry session between the router and the receiver. The session is established using either a dial-out mode or a dial-in mode, described in the Scale-Up Your Network Monitoring Strategy Using Telemetry article.

Note Watch this video to discover the power of real-time network management using model-driven telemetry.

This article describes the dial-out mode where the router dials out to the receiver to establish a telemetry session. In this mode, destinations and sensor-paths are configured and bound together into one or more subscriptions. The router continually attempts to establish a session with each destination in the subscription, and streams data to the receiver. The dial-out mode of subscriptions is persistent. Even when a session terminates, the router continually attempts to re-establish a new session with the receiver at regular intervals.

The following image shows a high-level overview of the dial-out mode:

Figure 5: Dial-Out Mode



This article describes, with a use case that illustrates the monitoring of CPU utilization, how streaming telemetry data helps you gain better visibility of your network, and make informed decisions to stabilize your network.

Tip You can programmatically configure a dial-out telemetry session using `openconfig-telemetry.yang` OpenConfig data model. To get started with using data models, see the *Programmability Configuration Guide*.

- Monitor CPU Utilization Using Telemetry Data to Plan Network Infrastructure, on page 20

Monitor CPU Utilization Using Telemetry Data to Plan Network Infrastructure

The use case illustrates how, with the dial-out mode, you can use telemetry data to proactively monitor CPU utilization. Monitoring CPU utilization ensures efficient storage capabilities in your network. This use case describes the tools used in the open-sourced collection stack to store and analyse telemetry data.

Note Watch this video to see how you configure model-driven telemetry to take advantage of data models, open source collectors, encodings and integrate into monitoring tools.

Telemetry involves the following workflow:

- **Define:** You define a subscription to stream data from the router to the receiver. To define a subscription, you create a destination-group and a sensor-group.
- **Deploy:** The router establishes a subscription-based telemetry session and streams data to the receiver. You verify subscription deployment on the router.
- **Operate:** You consume and analyse telemetry data using open-source tools, and take necessary actions based on the analysis.

Before you begin

Make sure you have L3 connectivity between the router and the receiver.

Define a Subscription to Stream Data from Router to Receiver

Create a subscription to define the data of interest to be streamed from the router to the destination.

Procedure

Step 1 Create one or more destinations to collect telemetry data from a router. Define a destination-group to contain the details about the destinations. Include the destination address (IPv4 or ipv6), or FQDN, port, transport, and encoding format in the destination-group.

Example:

Create a destination-group using data model

This example uses the native data model `Cisco-IOS-XR-um-telemetry-model-driven-cfg.yang`.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get-config>
    <source>
      <candidate/>
    </source>
    <filter>
      <telemetry-model-driven
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-telemetry-model-driven-cfg">
        <destination-groups>
          <destination-group>
            <destination-id>CPU-Health</destination-id>
            <ipv4-destinations>
              <ipv4-destination>
                <ipv4-address>172.0.0.0</ipv4-address>
                <destination-port>57500</destination-port>
                <encoding>self-describing-gpb</encoding>
                <protocol>
                  <protocol>tcp</protocol>
                </protocol>
              </ipv4-destination>
            </ipv4-destinations>
          </destination-group>
        </destination-groups>
      </telemetry-model-driven>
    </filter>
  </get-config>
</rpc>
```

Create a destination group using CLI

```
##Configuration with tls-hostname##
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group CPU-Health
Router(config-model-driven-dest)#address family ipv4 172.0.0.0 port 57500
Router(config-model-driven-dest-addr)#encoding self-describing-gpb
Router(config-model-driven-dest-addr)#protocol tcp
Router(config-model-driven-dest-addr)#commit
```

- `CPU-Health` is the name of the destination-group
- `172.0.0.0` is the IP address of the destination where data is to be streamed

  Note
  To avoid hard-coding IP address, the router can chose any of the configured IPv4 or ipv6 address using domain name service. If an established connections fails, the router connects to another resolved IP address, and streams data to that IP address.

- `57500` is the port number of the destination
- `self-describing-gpb` is the format in which data is encoded and streamed to the destination
- `tcp` is the protocol through which data is transported to the destination.

The destination for dial-out configuration supports IP address (Ipv4 or IPv6), and fully qualified domain name (FQDN) using domain name services (DNS). To use FQDN, you must assign IP address to the domain name.The domain name is limited to 128 characters. If DNS lookup fails for the provided domain name, the internal timer is activated for 30 sec. With this, the connectivity is continually tried every 30 sec until the domain named is looked-up successfully. DNS provides an address list depending on the address-family being requested. For example, on the router, the IP address for domain name is set using the following commands for IPv4 and ipv6 respectively:

```
domain ipv4 host abcd 172.x.x.1 172.x.x.2
```

```
domain ipv6 host abcd fd00:xx:xx:xx:1::1 fd00:xx:xx:xx:1::3
```

Step 2 Specify the subset of the data that you want to stream from the router using sensor paths. The sensor path represents the path in the hierarchy of a YANG data model. Create a sensor-group to contain the sensor paths.

Example:

Create a sensor-group for CPU utilization using data model

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <telemetry-model-driven
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-telemetry-model-driven-cfg">
        <sensor-groups>
          <sensor-group>
            <sensor-group-identifier>Monitor-CPU</sensor-group-identifier>
            <sensor-paths>
              <sensor-path>

<telemetry-sensor-path>Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization</telemetry-sensor-path>

              </sensor-path>
            </sensor-paths>
          </sensor-group>
        </sensor-groups>
      </telemetry-model-driven>
    </config>
  </edit-config>
</rpc>
```

Create a sensor-group for CPU utilization using CLI

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group Monitor-CPU
```

```
Router(config-model-driven-snsr-grp)# sensor-path
Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
Router(config-model-driven-snsr-grp)# commit
```

- `Monitor-CPU` is the name of the sensor-group
- `Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization` is the sensor path from where data is streamed.

Step 3 Subscribe to telemetry data that is streamed from a router. A subscription binds the destination-group with the sensor-group and sets the streaming method. The streaming method can be cadence-driven or event-driven telemetry.

Example:

Note
The configuration for event-driven telemetry is similar to cadence-driven telemetry, with only the sample interval as the differentiator. Configuring the sample interval value to `0`, zero, sets the subscription for event-driven telemetry, while configuring the interval to any non-zero value sets the subscription for cadence-driven telemetry.

Create a subscription using data model

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <telemetry-model-driven
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-telemetry-model-driven-cfg">
        <subscriptions>
          <subscription>
            <subscription-identifier>CPU-Utilization</subscription-identifier>
            <sensor-profiles>
              <sensor-profile>
                <sensorgroupid>Monitor-CPU</sensorgroupid>
                <sample-interval>30000</sample-interval>
              </sensor-profile>
            </sensor-profiles>
            <destination-profiles>
              <destination-profile>
                <destination-id>CPU-Health</destination-id>
              </destination-profile>
            </destination-profiles>
           <source-interface>Interface1</source-interface>
          </subscription>
        </subscriptions>
      </telemetry-model-driven>
    </config>
  </edit-config>
</rpc>
```

Create a subscription using CLI

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription CPU-Utilization
Router(config-model-driven-subs)#sensor-group-id Monitor-CPU sample-interval 30000
Router(config-model-driven-subs)#destination-id CPU-Health
Router(config-model-driven-subs)#source-interface Interface1
Router(config-model-driven-subs)#commit
```

where -

- `CPU-Utilization` is the name of the subscription
- `Monitor-CPU` is the name of the sensor-group
- `CPU-Health` is the name of the destination-group
- `Interface1` is the source interface that is used for establishing the telemetry session. If both the VRF and source interface are configured, the source interface must be in the same VRF as the one specified in the destination group.
- `30000` is the sample interval in milliseconds. The sample interval is the time interval between two streams of data. In this example, the sample interval is 30000 milliseconds or 30 seconds.

Verify Deployment of the Subscription

The router dials out to the receiver to establish a session with each destination in the subscription. After the session is established, the router streams data to the receiver to create a data lake.

You can verify the deployment of the subscription on the router.

Procedure

Step 1 View the model-driven telemetry configuration on the router.

Example:

```
Router#show running-config telemetry model-driven
telemetry model-driven
destination-group CPU-Health
address-family ipv4 172.0.0.0 port 57500
encoding self-describing-gpb
protocol tcp
!
sensor-group Monitor-CPU
sensor-path
Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
!
subscription CPU-Utilization
sensor-group-id Monitor-CPU sample-interval 30000
destination-id CPU-Health
source-interface GigabitEthernet0/0/0/0
 !
!
```

Step 2 Verify the state of the subscription. An `Active` state indicates that the router is ready to stream data to the receiver based on the subscription.

Example:

```
Router# show telemetry model-driven subscription CPU-Utilization

Subscription:  CPU-Utilization
-------------
  State:       NA
  Source Interface:       GigabitEthernet0_0_0_0( 0x0)
  Sensor groups:
```

```
Id: Monitor-CPU
  Sample Interval:      30000 ms
  Sensor Path:          Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
  Sensor Path State:    Resolved

Destination Groups:
Group Id: CPU-Health
  Destination IP:       172.0.0.0
  Destination Port:     57500
  Encoding:             self-describing-gpb
  Transport:            tcp
  State:                NA
  No TLS

Collection Groups:
------------------
No active collection groups
```

The router streams data to the receiver using the subscription-based telemetry session and creates a data lake in the receiver.

Step 3 Check for the error.

Example:

```
Router#show tech-support telemetry model-driven
Thu Nov 28 12:47:53.164 UTC
++ Show tech start time: 2024-Nov-28.124753.UTC ++
Thu Nov 28 12:47:54 UTC 2024 Waiting for gathering to complete
..
Thu Nov 28 12:48:00 UTC 2024 Compressing show tech output
Show tech output available at 0/RP0/CPU0 :
/harddisk:/showtech/showtech-telemetry_model_driven-2024-Nov-28.124753.UTC.tgz
++ Show tech end time: 2024-Nov-28.124800.UTC ++
```

Operate on Telemetry Data for In-depth Analysis of the Network

You can start consuming and analyzing telemetry data from the data lake using an open-sourced collection stack. This use case uses the following tools from the collection stack:

- Pipeline is a lightweight tool used to collect data. You can download Network Telemetry Pipeline from GitHub. You define how you want the collector to interact with routers and where you want to send the processed data using `pipeline.conf` file.
- Telegraph (plugin-driven server agent) and InfluxDB (a time series database (TSDB)) stores telemetry data, which is retrieved by visualization tools. You can download InfluxDB from GitHub. You define what data that you want to include into your TSDB using the `metrics.json` file.
- Grafana is a visualization tool that displays graphs and counters for data streamed from the router.

In summary, Pipeline accepts TCP and gRPC telemetry streams, converts data and pushes data to the InfluxDB database. Grafana uses the data from InfluxDB database to build dashboards and graphs. Pipeline and InfluxDB may run on the same server or on different servers.

Consider that the router is streaming data of approximately 350 counters every 5 seconds, and Telegraf requests information from the Pipeline at 1-second intervals. The CPU usage is analyzed in three stages using:

- a single router to get initial values
- two routers to find the difference in values and understand the pattern.
- five routers to arrive at a proof-based conclusion.

This helps you make informed business decisions about deploying the infrastructure; in this case, the CPU.

Procedure

Step 1 Start Pipeline, and enter your router credentials.

Note
The IP address and port that you specify in the destination-group must match the IP address and port on which Pipeline is listening.

Example:

```
$ bin/pipeline -config pipeline.conf

Startup pipeline
Load config from [pipeline.conf], logging in [pipeline.log]

CRYPT Client [grpc_in_mymdtrouter], [http://172.0.0.0:5432]
 Enter username: <username>
 Enter password: <password>
Wait for ^C to shutdown
```

Step 2 In the Telegraph configuration file, add the following values to read the metrics about CPU usage.

Example:

```
[[inputs.cpu]]
  ## Whether to report per-cpu stats or not
  percpu = true
  ## Whether to report total system cpu stats or not
  totalcpu = true
  ## If true, collect raw CPU time metrics.
  collect_cpu_time = false
  ## If true, compute and report the sum of all non-idle CPU states.
  report_active = false
```

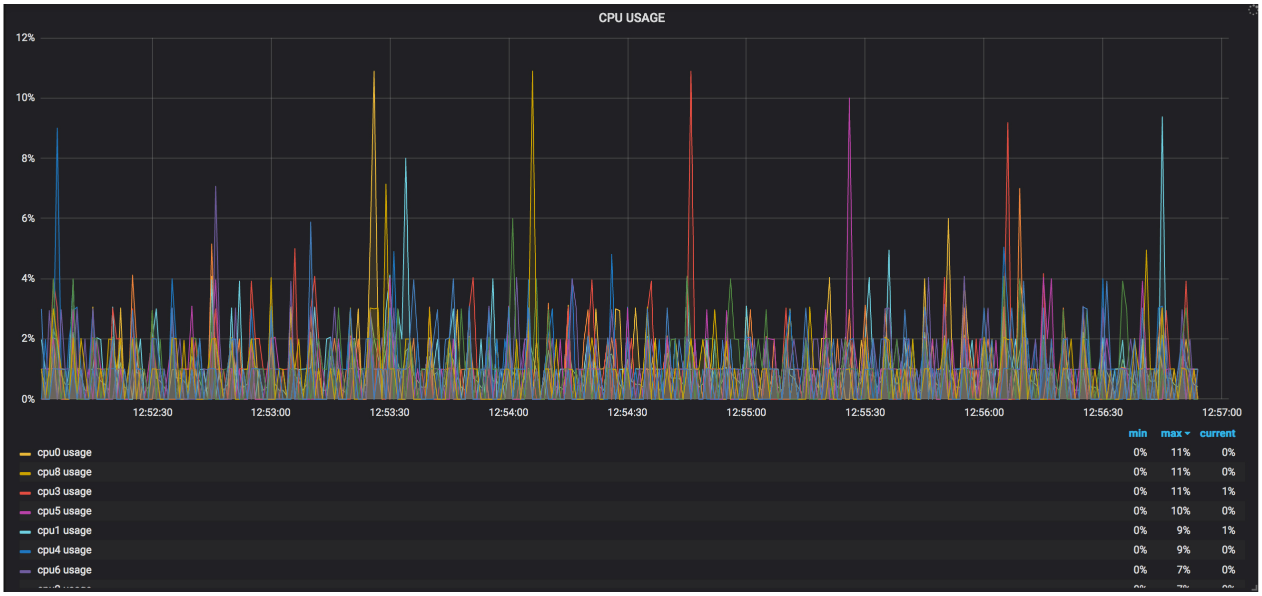
Step 3 Use Grafana to create a dashboard and visualize data about CPU usage.

One router

The router pushes the counters every five seconds.

All CPU cores are loaded equally, and there are spikes up to approximately 10 or 11 percent.

Figure 6: CPU Usage Graph with a Single Router

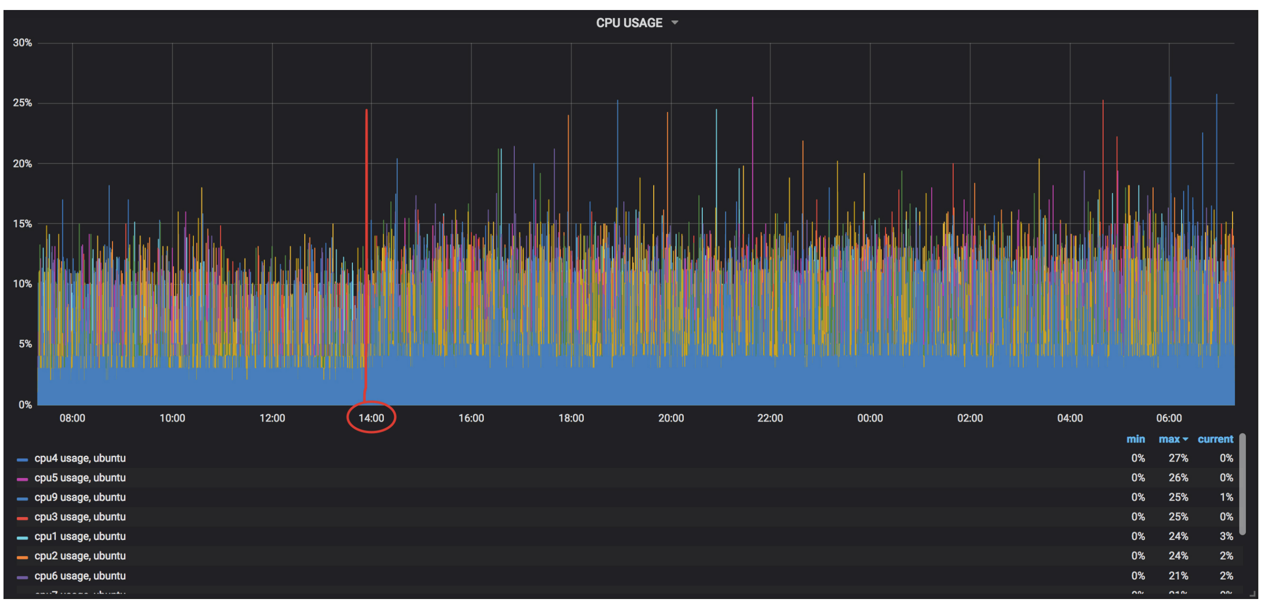


369756

Two routers

The second router is added at `14:00` in the timeline, and shows an increase in the spikes to around 25 percent with the midpoint value at 15 percent.

Figure 7: CPU Usage Graph with Two Routers

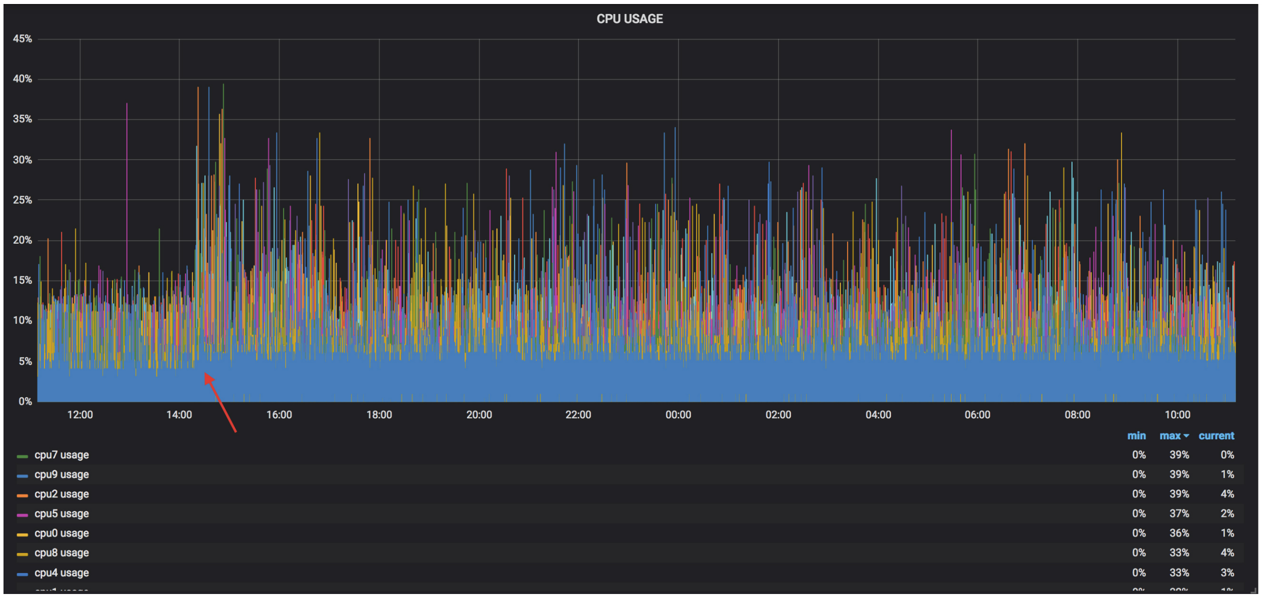


369757

Five routers

With five routers, the spikes peak up to approximately 40 percent with the midpoint in the range of 22 to 25 percent.

Figure 8: CPU Usage Graph with Five Routers



369755

In conclusion, telemetry data shows that the processes are balanced almost equally across the CPU cores. There is no linear increase on a subset of cores. This analysis helps in planning the CPU utilization based on the number of counters that you stream.

CHAPTER 5

Establish a Model-Driven Telemetry Session from a Collector to a Router

Streaming telemetry is a new paradigm in monitoring the health of a network. It provides a mechanism to efficiently stream configuration and operational data of interest from Cisco IOS XR routers. This streamed data is transmitted in a structured format to remote management stations for monitoring and troubleshooting purposes.

With telemetry data, you create a data lake. Analyzing this data, you proactively monitor your network, monitor utilization of CPU and memory, identify patterns, troubleshoot your network in a predictive manner, and devise strategies to create a resilient network using automation.

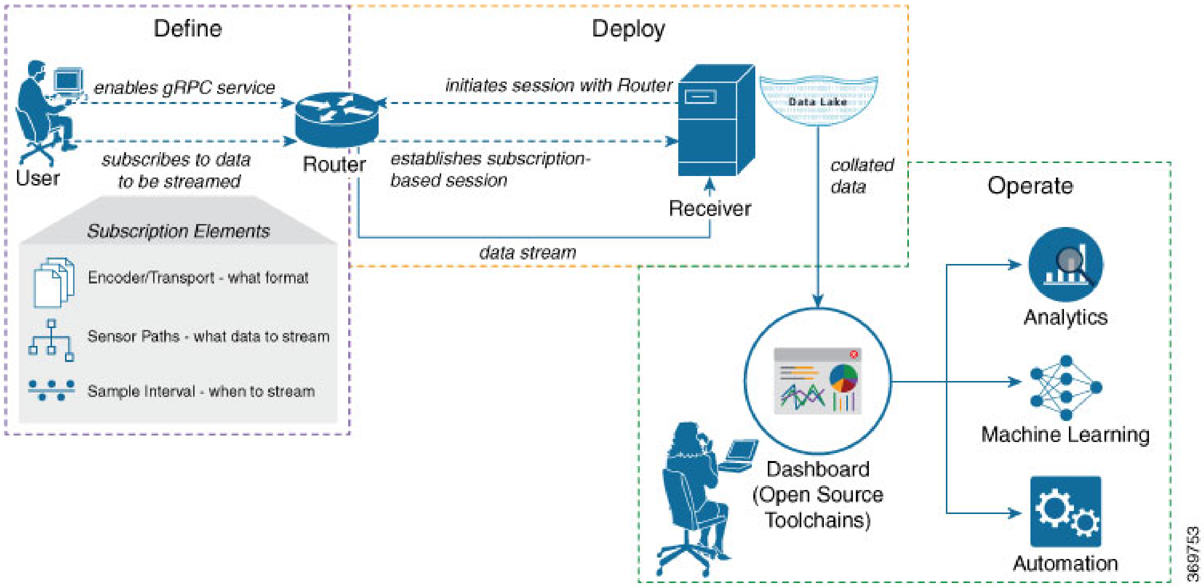
Telemetry works on a subscription model where you subscribe to the data of interest in the form of sensor paths. The sensor paths describes OpenConfig data models or native Cisco data models. You can access the OpenConfig and Native data models for telemetry from Github, a software development platform that provides hosting services for version control. You choose who initiates the subscription by establishing a telemetry session between the router and the receiver. The session is established using either a either a dial-out mode or dial-in mode, described in the Scale-Up Your Network Monitoring Strategy Using Telemetry article.

Note Watch this video to discover the power of real-time network management using model-driven telemetry.

This article describes the dial-in mode where a receiver dials in to the router to establish a telemetry session. In this mode, the receiver dials in to the router, and subscribes dynamically to one or more sensor paths specified in a subscription. The router streams telemetry data through the same session that is established by the receiver. The dial-in mode of subscriptions is dynamic. This dynamic subscription terminates when the receiver cancels the subscription or when the session terminates.

The following image shows a high-level overview of the dial-in mode:

Figure 9: Dial-In Mode



This article describes, with a use case that illustrates the simultaneous monitoring of various parameters in the network, how streaming telemetry data helps you gain better visibility of the network, and make informed decisions to stabilize it.

YANG Data Model You can programmatically configure a dial-in telemetry session using openconfig-telemetry.yang OpenConfig data model. To get started with using data models, see the *Programmability Configuration Guide*.

• Monitor Network Parameters Using Telemetry Data for Proactive Analysis, on page 30

Monitor Network Parameters Using Telemetry Data for Proactive Analysis

The use case illustrates how, with the dial-in mode, you can use telemetry data to stream various parameters about your network. You use this data for predictive analysis where you monitor patterns, and proactively troubleshoot issues. This use case describes the tools used in the open-sourced collection stack to store and analyse telemetry data.

Note Watch this video to see how you configure model-driven telemetry to take advantage of data models, open source collectors, encodings and integrate into monitoring tools.

Telemetry involves the following workflow:

- **Define:** You define a subscription to stream data from the router to the receiver. To define a subscription, you create a sensor-group.

- **Deploy:** The receiver initiates a session with the router and establishes a subscription-based telemetry session. The router streams data to the receiver. You verify subscription deployment on the router.
- **Operate:** You consume and analyse telemetry data using open-source tools, and take necessary actions based on the analysis.

Before you begin

Ensure you meet these dependancies:

- Make sure you have L3 connectivity between the router and the receiver.
- Enable gRPC server on the router to accept incoming connections from the receiver.

```
Router#configure
Router(config)#grpc
Router(config-grpc)#port <port-number>
Router(config-grpc)#commit
```

The port-number ranges from 57344 to 57999. If a port number is unavailable, an error is displayed.

Define a Subscription to Stream Data from Router to Receiver

Create a subscription to define the data of interest to be streamed from the router to the destination.

Procedure

Step 1 Specify the subset of the data that you want to stream from the router using sensor paths. The sensor path represents the path in the hierarchy of a YANG data model. This example uses the native data model `Cisco-IOS-XR-um-telemetry-model-driven-cfg.yang`. Create a sensor-group to contain the sensor paths.

Example:

```
sensor-group health
  sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
  sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary
  sensor-path Cisco-IOS-XR-shellutil-oper:system-time/uptime
 !
 sensor-group interfaces
  sensor-path
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters
  sensor-path Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-summary
 !
 sensor-group optics
  sensor-path Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info
 !
 sensor-group routing
  sensor-path Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/levels/level/adjacencies/adjacency

  sensor-path Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/statistics-global
  sensor-path
Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ip-rib-route-table-name/protocol/isis/as/information

  sensor-path Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance/instance-active/default-vrf/process-info

 !
```

```
 sensor-group mpls-te
 sensor-path Cisco-IOS-XR-mpls-te-oper:mpls-te/tunnels/summary
 sensor-path Cisco-IOS-XR-ip-rsvp-oper:rsvp/interface-briefs/interface-brief
 sensor-path Cisco-IOS-XR-ip-rsvp-oper:rsvp/counters/interface-messages/interface-message
 !
```

Step 2 Subscribe to telemetry data that is streamed from a router. A subscription binds the sensor-group, and sets the streaming method. The streaming method can be cadence-driven or event-driven. Seperating the sensor-paths into different subscriptions enhances the efficiency of the router to retrieve operational data at scale.

Example:

Note
The configuration for event-driven telemetry is similar to cadence-driven telemetry, with only the sample interval as the differentiator. Configuring the sample interval value to `0` (zero), sets the subscription for event-driven telemetry, while configuring the interval to any non-zero value sets the subscription for cadence-driven telemetry.

```
subscription health
  sensor-group-id health strict-timer
  sensor-group-id health sample-interval 30000
 !
 subscription interfaces
  sensor-group-id interfaces strict-timer
  sensor-group-id interfaces sample-interval 30000
 !
 subscription optics
  sensor-group-id optics strict-timer
  sensor-group-id optics sample-interval 30000
 !
 subscription routing
  sensor-group-id routing strict-timer
  sensor-group-id routing sample-interval 30000
 !
 subscription mpls-te
  sensor-group-id mpls-te strict-timer
  sensor-group-id mpls-te sample-interval 30000
 !
```

Verify Deployment of the Subscription

The receiver dials into the router to establish a dynamic session based on the subscription. After the session is established, the router streams data to the receiver to create a data lake.

You can verify the deployment of the subscription on the router.

Procedure

Verify the state of the subscription. An `Active` state indicates that the router is ready to stream data to the receiver based on the subscription.

Example:

```
Router#show telemetry model-driven subscription
Thu Jan  16 09:48:14.293 UTC
```

```
Subscription:  health                    State: Active
-------------
  Sensor groups:
  Id                                 Interval(ms)        State
  health                             30000               Resolved

Subscription:  optics                    State: NA
-------------
  Sensor groups:
  Id                                 Interval(ms)        State
  optics                             30000               Resolved

Subscription:  mpls-te                   State: NA
-------------
  Sensor groups:
  Id                                 Interval(ms)        State
  mpls-te                            30000               Resolved

Subscription:  routing                   State: NA
-------------
  Sensor groups:
  Id                                 Interval(ms)        State
  routing                            30000               Resolved

Subscription:  interfaces                State: NA
-------------
  Sensor groups:
  Id                                 Interval(ms)        State
  interfaces                         30000               Resolved

Subscription:  CPU-Utilization           State: NA
-------------
  Sensor groups:
  Id                                 Interval(ms)        State
  Monitor-CPU                        30000               Resolved

  Destination Groups:
  Id                 Encoding            Transport   State   Port    Vrf     IP
  CPU-Health         self-describing-gpb tcp         NA      57500           172.0.0.0
    No TLS
```

The router streams data to the receiver using the subscription-based telemetry session and creates a data lake in the receiver.

Operate on Telemetry Data for In-depth Analysis of the Network

You can start consuming and analyzing telemetry data from the data lake using an open-sourced collection stack. This use case uses the following tools from the collection stack:

- Pipeline is a lightweight tool used to collect data. You can download Network Telemetry Pipeline from Github. You define how you want the collector to interact with routers, and where you want to send the processed data using `pipeline.conf` file.
- Telegraph or InfluxDB is a time series database (TSDB) that stores telemetry data, which is retrieved by visualization tools. You can download InfluxDB from Github. You define what data you want to include into your TSDB using the `metrics.json` file.
- Grafana is a visualization tool that displays graphs and counters for data streamed from the router.

In summary, Pipeline accepts TCP and gRPC telemetry streams, converts data and pushes data to the InfluxDB database. Grafana uses the data from InfluxDB database to build dashboards and graphs. Pipeline and InfluxDB may run on the same server or on different servers.

Consider that the router is monitored for the following parameters:

- Memory and CPU utilization
- Interface counters and interface summary
- Transmitter and receiver power levels from optic controllers
- ISIS route counts and ISIS interfaces
- BGP neighbours, path count, and prefix count
- MPLS-TE tunnel summary
- RSVP control messages and bandwidth allocation for each interface

Procedure

Step 1 Start Pipeline from the shell, and enter your router credentials.

Example:

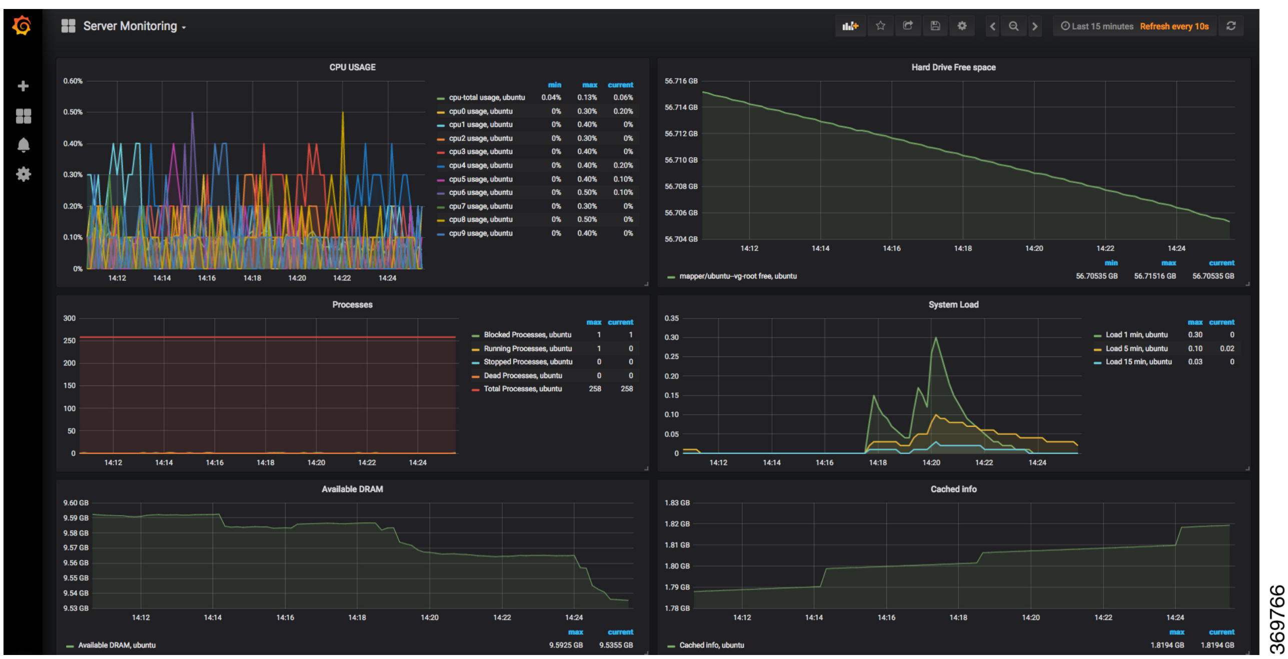
```
$ bin/pipeline -config pipeline.conf

Startup pipeline
Load config from [pipeline.conf], logging in [pipeline.log]

CRYPT Client [grpc_in_mymdtrouter], [http://172.0.0.0:5432]
 Enter username: <username>
 Enter password: <password>
Wait for ^C to shutdown
```

The streamed telemetry data is stored in InfluxDB.

Step 2 Use Grafana to create a dashboard and visualize the streamed data.

Figure 10: Visual Analysis of Network Health using Telemetry Data

Figure 11: Visual Analysis of System Monitoring using Telemetry Data



In conclusion, telemetry data shows that various parameters of the network can be monitored simulatanously. This data is streamed in near real-time without affecting the performance of the network. With this data, you gain better visibility into your network.

CHAPTER 6

Enhancements to Streaming Telemetry

This section provides an overview of the enhancements made to streaming telemetry data.

Hardware Timestamp

Table 5: Feature History Table

Feature Name	Release Information	Description
Enhancements to Hardware Timestamp	Release 7.3.4	Telemetry messages carry a timestamp per interface to indicate the time when data is collected from the hardware. With this feature, the support for hardware timestamp is extended to MPLS Traffic Engineering (MPLS TE) counters, Segment Routing for Traffic Engineering (SR-TE) interface counters, protocol statistics, and bundle protocol counters. The interface counters in the following paths are enhanced for hardware timestamp: • Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters • Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters • openconfig-network-instance:network-instances/network-instance/mpls/lsps/constrained-path/tunnels • openconfig-interfaces:interfaces/interface
Hardware Timestamp	Release 7.3.1	Whenever periodic statistics are streamed, the collector reads the data from its internal cache, instead of fetching the data from the hardware. When the data is read from the cache, the rate at which data is processed shows spikes because the timestamp from the collector is off by several seconds. With hardware timestamping, the inconsistencies that are observed when reading data from the cache file is removed.

Whenever periodic stats are streamed, the collector reads the stats from its internal cache, instead of fetching the stats from the hardware. When the data is read from the sensor paths of Stats manager cache, the rate calculation shows spikes. This behavior is due to the timestamp from the collector that is off by several seconds.

Therefore, timestamp of some other collector takes precedence because timestamps of collectors are not in synchronization with the current timestamp. This is observed when there are multiple collectors providing stats updates for the same interface.

The YANG data model for Stats manager `Cisco-IOS-XR-infra-statsd-oper.yang` is enhanced to enable the collector to read periodic stats data from the router using hardware timestamp.

The hardware timestamp is taken into account when a primary collector (for generic or proto stats) provides stats updates from the hardware to the Stats manager. With hardware timestamping in rate computation while streaming periodic stats, the spikes due to the timestamp issue is resolved.

The hardware timestamp is updated only when the collector attempts to read the counters from hardware. Else, the value remains `0`. The latest stats can be streamed at a minimum cadence of 10 seconds and periodic stats at a cadence of 30 seconds. The support is available only for physical interfaces and subinterfaces, and bundle interface and subinterfaces.

When there is no traffic flow on protocols for an interface, the hardware timestamp for the protocols is published as 0. This is due to non-synchronized timestamps sent by the collector for protocols in traffic as compared to non-traffic scenarios.

A non-zero value is published for protocols that have stats published by a primary collector for both traffic and non-traffic scenarios.

Note The hardware timestamp is supported only for primary collectors. When the hardware has no update, the timestamp will be same. However generic counters are computed for primary and non-primary collectors. The non-primary collectors show the latest stats, but not the timestamp.

When the counters are cleared for an interface using **clear counters interface** command, all counter-related data including the timestamps for the interface is cleared. After all counter values are cleared and set to `0`, the last data time is updated only when there is a request for it from a collector. For example, last data time gets updated from a collector:

```
Router#:Aug  7 09:01:08.471 UTC: statsd_manager_l[168]: Updated last data time for ifhandle
 0x02000408,
stats type 2 from collector with node 0x100, JID 250, last data time 1596790868.
INPUT: last 4294967295 updated 1596469986. OUTPUT: last 4294967295 updated 1596469986
```

All other counter values and hardware timestamp are updated when the counters are fetched from the hardware. In this case, all counters including the hardware timestamp is 0:


```
{"node_id_str":"MGBL_MTB_5504","subscription_id_str":"app_TEST_200000001",
"encoding_path":"Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters",
"collection_id":"7848",
"collection_start_time":"1596790879567",
"msg_timestamp":"1596790879571","data_json":
[{"timestamp":"1596790879570","keys":[{"interface-name":"FortyGigE0/1/0/11"}],
"content":{"packets-received":"0","bytes-received":"0","packets-sent":"0",
"bytes-sent":"0","multicast-packets-received":"0","broadcast-packets-received":"0",
"multicast-packets-sent":"0","broadcast-packets-sent":"0","output-drops":0,"output-queue-drops":0,
"input-drops":0,"input-queue-drops":0,"runt-packets-received":0,"giant-packets-received":0,
"throttled-packets-received":0,"parity-packets-received":0,"unknown-protocol-packets-received":0,
"input-errors":0,"crc-errors":0,"input-overruns":0,"framing-errors-received":0,"input-ignored-packets":0,
"input-aborts":0,"output-errors":0,"output-underruns":0,"output-buffer-failures":0,"output-buffers-swapped-out":0,
"applique":0,"resets":0,"carrier-transitions":0,"availability-flag":0,
"last-data-time":"1596790868","hardware-timestamp":"0",
"seconds-since-last-clear-counters":15,"last-discontinuity-time":1596469946,"seconds-since-packet-received":0,
"seconds-since-packet-sent":0}}],"collection_end_time":"1596790879571"}
```

Stream QoS Statistics Telemetry Data

Table 6: Feature History Table

Feature Name	Release Information	Description
Stream QoS Statistics Telemetry Data	Release 7.3.3	You can use the `Cisco-IOS-XR-qos-ma-oper.yang` data model to stream telemetry data on QoS statistics from the route processor (RP). The bundle statistics are now stored in the RP, where data is persistent, and its retrieval is unaffected by bundle member or line card failure. In earlier releases, QoS statistics was stored on line cards, and any bundle member or line card failure caused loss of statistics data.

You can collect QoS statistics for physical, virtual, bundle interfaces and subinterfaces using a push mechanism where data is streamed out of the router at a cadence. You can also collect data about ingress, egress policy-map statistics and VoQ statistics of an egress policy-map. When data is collected from the hardware, the statistics data is time-stamped.

To enable the feature, use the **hw-module profile qos qos-stats-push-collection** command in XR Config mode. You must reload the router for the configuration to take effect. To clear the QoS statistics on an interface, use **clear qos counters interface** *interface name* command in XR Exec mode. To clear the statistics for all interfaces, use **clear qos counters interface all** command.

Note When the counters are cleared, the counters for SNMP statistics are also cleared.

For more information on modular QoS on link bundles, see *Modular QoS Command Reference for Cisco 8000 Series Routers*.

With this release, the interface statistics can be displayed using `Cisco-IOS-XR-qos-ma-oper.yang` data model. You can stream telemetry data from the sensor path:

```
Cisco-IOS-XR-qos-ma-oper:qos/interface-table/interface/input/service-policy-names/service-policy-instance/statistics

Cisco-IOS-XR-qos-ma-oper:qos/interface-table/interface/output/service-policy-names/service-policy-instance/statistics

Cisco-IOS-XR-qos-ma-oper:qos/interface-table/interface[interface-name=<interface-name>]/input/service-policy-names/service-policy-instance[service-policy-name=egress]/statistics

Cisco-IOS-XR-qos-ma-oper:qos/interface-table/interface[interface-name=<interface-name>]/output/service-policy-names/service-policy-instance[service-policy-name=egress]/statistics
```

> **Note** This feature is not supported for all rate counters like matched-rate, transmitted-rate, dropped-rate in the QoS statistics. The bundle member statistics, location-based SPI statistics, sensor path for bundle members statistics, are not supported.

Usage Guidelines and Limitations

The feature has a limitation where only one type of QoS egress statistics, either marking or queuing, is available when multi-policy maps are configured on a bundle interface using the **show operational QoS InterfaceTable Interface/InterfaceName=Bundle-Ether100 Output Statistics** CLI command.

To view both egress statistics, you must use the alternative command **show operational QoS InterfaceTable Interface/InterfaceName=Bundle-Ether100 Output**.

The following steps show the configuration to stream data about bundle statistics to the collector.

Procedure

Step 1 Configure QoS on link bundles.

Example:

QoS Profile:

```
Router(config)# hw-module profile qos qos-stats-push-collection
Wed Dec 22 06:35:48.251 UTC
In order to activate this new qos profile, you must manually reload the chassis/all line cards
Router(config)#commit
```

Class-map:

```
Router(config)#class-map TC3
Router(config-cmap)#match traffic-class 1
Router(config-cmap)#commit
```

Policy-map:

```
Router(config)#policy-map egress
Router(config-pmap)#class TC3
Router(config-pmap-c)#shape average 1 mbps
Router(config-pmap-c)#commit
```

Step 2 Attach the service policy to an interface.

Example:

```
Router(config)#int hundredGigE 0/0/1/0.5
Router(config-if)#service-policy output egress
Router(config-if)#commit
```

Step 3 Configure telemetry subscription.

Example:

```
Router#show run telemetry model-driven
Wed Dec 22 10:27:10.846 UTC
telemetry model-driven
 destination-group destination-4
  address-family ipv4 5.16.4.114 port 51023
   encoding json
```

```
    protocol grpc no-tls
   !
  !
  sensor-group qos-grp
   sensor-path Cisco-IOS-XR-qos-ma-oper:qos/interface-table/interface
   [interface-name=HundredGigE0/0/1/0.5]/output/service-policy-names/service-policy-instance
   [service-policy-name=egress]/statistics
  !
 subscription qos-subs
   sensor-group-id qos-grp sample-interval 10000
   destination-id destination-4
  !
 !
```

Verification

Verify the QoS telemetry configuration.

Policy Map:

```
Router#show policy-map pmap-name egress detail
Wed Dec 22 06:50:53.779 UTC
class-map match-any TC3
 match traffic-class 1
 end-class-map
!
policy-map egress
 class TC3
  shape average 1 mbps
 !
 class class-default
 !
 end-policy-map
!
```

Statistics Data:

```
Router#show policy-map interface HundredGigE0/0/1/0.5 output
Wed Dec 22 08:29:18.603 UTC

HundredGigE0/0/1/0.5 output: egress

Class TC3
  Classification statistics          (packets/bytes)     (rate - kbps)
    Matched             :               55563/55563000            0
    Transmitted         :               55401/55401000            0
    Total Dropped       :                 162/162000              0
  Queueing statistics
    Queue ID                             : 1377
    High watermark  (Unknown)
    Inst-queue-len  (Unknown)
    Avg-queue-len   (Unknown)
    Taildropped(packets/bytes)           : 162/162000
    Queue(conform)      :                    0/0                  0
    Queue(exceed)       :                    0/0                  0
Class class-default
  Classification statistics          (packets/bytes)     (rate - kbps)
    Matched             :             1077710/1077710000          0
    Transmitted         :             1077710/1077710000          0
    Total Dropped       :                    0/0                  0
  Queueing statistics
```

```
    Queue ID                             : 1376
    High watermark  (Unknown)
    Inst-queue-len  (Unknown)
    Avg-queue-len   (Unknown)
    Taildropped(packets/bytes)           : 0/0
    Queue(conform)      :                  0/0                    0
    Queue(exceed)       :                  0/0                    0
Policy Bag Stats time: 1637137751027  [Local Time: 12/22/21 08:29:11.027]
```

QoS Statistics Using Telemetry:

The sample output shows the telemetry data streamed from the router:


```
{
    "node_id_str": "R1",
    "subscription_id_str": "qos-subs",
    "encoding_path":
"Cisco-IOS-XR-qos-ma-oper:qos/interface-table/interface/output/service-policy-names/service-policy-instance/statistics",

    "collection_id": "21226",
    "collection_start_time": "1637144915201",
    "msg_timestamp": "1637144915203",
    "data_json": [
        {
            "timestamp": "1637144891032",
            "keys": [
                {
                    "interface-name": "HundredGigE0/0/1/0.5"
                },
                {
                    "service-policy-name": "egress"
                }
            ],
            "content": {
                "policy-name": "egress",
                "state": "active",
                "class-stats": [
                    {
                        "counter-validity-bitmask": "270532608",
                        "class-name": "TC3",
                        "cac-state": "unknown",
                        "general-stats": {
                            "transmit-packets": "239519",
                            "transmit-bytes": "239519000",
                            "total-drop-packets": "798",
                            "total-drop-bytes": "798000",
                            "total-drop-rate": 0,
                            "match-data-rate": 0,
                            "total-transmit-rate": 0,
                            "pre-policy-matched-packets": "240317",
                            "pre-policy-matched-bytes": "240317000"
                        },
                        "queue-stats-array": [
                            {
                                "queue-id": 1377,
                                "tail-drop-packets": "798",
                                "tail-drop-bytes": "798000",
                                "queue-drop-threshold": 0
"forced-wred-stats-display": false,
                                "random-drop-packets": "0",
                                "random-drop-bytes": "0",
                                "max-threshold-packets": "0",
                                "max-threshold-bytes": "0",
                                "conform-packets": "0",
                                "conform-bytes": "0",
```

```
                                    "exceed-packets": "0",
                                    "exceed-bytes": "0",
                                    "conform-rate": 0,
                                    "exceed-rate": 0
                                }
                            ]
                        },
                        {
                            "counter-validity-bitmask": "270532608",
                            "class-name": "class-default",
                            "cac-state": "unknown",
                            "general-stats": {
                                "transmit-packets": "4661952",
                                "transmit-bytes": "4661952000",
                                "total-drop-packets": "0",
                                "total-drop-bytes": "0",
                                "total-drop-rate": 0,
                                "match-data-rate": 0,
                                "total-transmit-rate": 0,
                                "pre-policy-matched-packets": "4661952",
                                "pre-policy-matched-bytes": "4661952000"
                            },
 "queue-stats-array": [
                                {
                                    "queue-id": 1376,
                                    "tail-drop-packets": "0",
                                    "tail-drop-bytes": "0",
                                    "queue-drop-threshold": 0,
                                    "forced-wred-stats-display": false,
                                    "random-drop-packets": "0",
                                    "random-drop-bytes": "0",
                                    "max-threshold-packets": "0",
                                    "max-threshold-bytes": "0",
                                    "conform-packets": "0",
                                    "conform-bytes": "0",
                                    "exceed-packets": "0",
                                    "exceed-bytes": "0",
"conform-rate": 0,
                                    "exceed-rate": 0
                                }
                            ]
                        }
                    ],
                    "satid": 0,
                    "policy-timestamp": "1637144891032"
                }
            }
        ],
        "collection_end_time": "1637144915203"
}
```

Enhanced Syslog Notifications for Unresolved Line Card Forwarding Paths

Table 7: Feature History Table

Feature Name	Release Information	Description
Enhanced Syslog Notifications for Unresolved Line Card Forwarding Paths	Release 7.5.2Release 7.3.3	This feature notifies you of Line Card and Route Processor paths not resolving in the Forwarding Information Base. Both Model-Driven Telemetry (MDT) and Event Driven Telemetry (EDT) notifications are supported. In earlier releases, notifications for route processors were supported. This feature provides for improved diagnostics.

Telemetry now supports syslog notification from line cards. This is in addition to the existing notification support from route processors. You will be notified of line card and route processor paths not resolving in the Forwarding Information Base (FIB), through MDT and EDT notifications.

MDT is configured for cadence-based telemetry, while EDT is configured for event-based notification. Notifications are generated only when the device goes into error or OOR state, and during device recovery. Errors and OOR are tracked for a device as a whole, and not for individual nodes. The IPv4 Error, IPv6 Error, IPv4 OOR, and IPv6 OOR telemetry notifications are supported.

The following notification is an example of IPv4 error state, if a line card and route processor paths do not resolve in the FIB:

```
GPB(common) Message
[5.13.9.177:38418(PE1)/Cisco-IOS-XR-fib-common-oper:oc-aft-l3/protocol/ipv4/error/state msg
 len: 168]
{
    "Source": "5.13.9.177:38418",
    "Telemetry": {
        "node_id_str": "PE1",
        "subscription_id_str": "Sub2",
       "encoding_path": "Cisco-IOS-XR-fib-common-oper:oc-aft-l3/protocol/ipv4/error/state",

        "collection_id": 243,
        "collection_start_time": 1637858634881,
        "msg_timestamp": 1637858634881,
        "collection_end_time": 1637858634883
    },
    "Rows": [
        {
            "Timestamp": 1637858634882,
            "Keys": null,
            "Content": {
                "is-in-error-state": "true"
            }
        }
```

```
    ]
}
```


> **Note** The parameters denote`"Content": {"is-in-error-state": "true"}` that the system is in error state.

The following notification is an example of IPv4 OOR, if a line card and route processor are in OOR state:


```
GPB(common) Message
[5.13.9.177:50146(PE1)/Cisco-IOS-XR-fib-common-oper:oc-aft-l3/protocol/ipv4/oor/state msg
len: 163]
{
    "Source": "5.13.9.177:50146",
    "Telemetry": {
        "node_id_str": "PE1",
        "subscription_id_str": "Sub1",
        "encoding_path": "Cisco-IOS-XR-fib-common-oper:oc-aft-l3/protocol/ipv4/oor/state",
        "collection_id": 11,
        "collection_start_time": 1637815892624,
        "msg_timestamp": 1637815892624,
        "collection_end_time": 1637815892626
    },
    "Rows": [
        {
            "Timestamp": 1637815892625,
            "Keys": null,
            "Content": {
                "is-in-oor-state": "true"
            }
        }
    ]
}
```


> **Note** The parameters denote`"Content": {"is-in-oor-state": "true"}` that the system is in OOR state.

Target-Defined Mode for Cached Generic Counters Data

Table 8: Feature History Table

Feature Name	Release Information	Description
Target-Defined Mode for Cached Generic Counters Data	Release 7.5.1 Release 7.3.3	This feature streams telemetry data for cached generic counters using a TARGET_DEFINED subscription. This subscription ensures that any change to the cache streams the latest data to the collector as an event-driven telemetry notification. This feature introduces support for the following sensor path: `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters`

Streaming telemetry pushes the subscribed data from the router to one or more collectors. The telemetry infrastructure retrieves the data from the system database when you send a subscription request. Based on the subscription request or the telemetry configuration the cached generic counters data can be retrieved periodically based on the sample-interval. Data, such as interface statistics, is cached and refreshed at certain intervals. The `TARGET_DEFINED` subscription mode can be used to retrieve data when the cache gets updated, and is not based on a timer.

The application can register as a data producer with the telemetry library and the SysdB paths it supports. One of the data producers, Statsd, uses the library with a `TARGET_DEFINED` subscription mode. As part of this mode, the producer registers the sensor paths. The statistics infrastructure streams the incremental updates for statsd cache sensor path `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters`. With this path in the subscription, whenever cache is updated, the statsd application pushes the updates to the telemetry daemon. The daemon sends these incremental updates to the collector. The cache updates are pushed for physical interfaces, physical subinterfaces, bundle interfaces, and bundle subinterfaces. You can subscribe to the sensor path for the cached generic counters with `TARGET_DEFINED` mode instead of the sensor path for the latest generic counters (`Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters`) to reduce the system load.

Configure the router to stream telemetry data from cache for generic counters using the following instructions:

Create a `TARGET_DEFINED` subscription mode for cached generic counters using one of the two options:

- **Option 1:** gRPC Network Management Interface (gNMI) subscribe request

```
{
    "name": "SubscribeRequest",
    "subscribe": {
        "prefix": {"origin":
                "Cisco-IOS-XR-infra-statsd-oper"

              },
         "mode": "STREAM", "encoding": "PROTO", "updates_only": "false",
        "subscription": [
         { "path": {"elem": [ {"name":"infra-statistics"},
                             {"name":"interfaces"},
                             {"name":"interface"},
                             {"name":"cache"},
                             {"name":"generic-counters"}
                          ]
                },
           "mode":   "TARGET_DEFINED"
         }
                    ]
    }
}
```

- **Option 2:** Model-driven telemetry configuration for non-gNMI requests

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription sub1
Router(config-model-driven-subs)#sensor-group-id grp1 mode target-defined
Router(config-model-driven-subs)#source-interface Interface1
Router(config-model-driven-subs)#commit
```

After the subscription is triggered, updates to the stats cache are monitored. The statsd application pushes the cached generic counters to the client (collector).

View the number of incremental updates for the sensor path.

```
Router#show telemetry model-driven subscription .*
Fri Nov 12 23:36:27.212 UTC
Subscription: GNMI__16489080148754121540
-------------
   Collection Groups:
    ------------------
      Id: 1
      Sample Interval:      0 ms   (Incremental Updates)
      Heartbeat Interval:   NA
      Heartbeat always:     False
      Encoding:             gnmi-proto
      Num of collection:    1
      Incremental updates:  12
      Collection time:      Min:     5 ms Max:     5 ms
      Total time:           Min:     6 ms Avg:     6 ms Max:     6 ms
      Total Deferred:       1
      Total Send Errors:    0
      Total Send Drops:     0
      Total Other Errors:   0
      No data Instances:    0
      Last Collection Start:2021-11-12
                23:34:27.1362538876 +0000
      Last Collection End:  2021-11-12   23:34:27.1362545589
              +0000
      Sensor Path:          Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/
                            interface/cache/generic-counters
```

In this example, the incremental updates of `12` indicates that the cache is updated 12 times.

You can also retrieve the detailed operational data about the subscription using the following command. In this example, `statsd-target` is the subscription name.

```
Router#show telemetry model-driven subscription statsd-target internal
Fri Nov 12 08:51:16.728 UTC
Subscription: statsd-target
-------------
State: ACTIVE
Sensor groups:
Id: statsd
Sample Interval: 0 ms (Incremental Updates)
Heartbeat Interval: NA
Sensor Path: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/
             generic-counters
Sensor Path State: Resolved

Destination Groups:
Group Id: statsd-target
Destination IP: 192.0.2.1
Destination Port: 56000
Encoding: json
Transport: grpc
State: Active
TLS : False
Total bytes sent: 623656
Total packets sent: 13
Last Sent time: 2021-08-16 08:51:15.1304821089 +0000

Collection Groups:
------------------
Id: 2
Sample Interval: 0 ms (Incremental Updates)
Heartbeat Interval: NA
Heartbeat always: False
```

```
Encoding: json
Num of collection: 1
Incremental updates: 3
Collection time: Min: 94 ms Max: 94 ms
Total time: Min: 100 ms Avg: 100 ms Max: 100 ms
Total Deferred: 0
Total Send Errors: 0
Total Send Drops: 0
Total Other Errors: 0
No data Instances: 0
Last Collection Start:2021-08-16 08:51:04.1293895665 +0000
Last Collection End: 2021-08-16 08:51:04.1293996284 +0000
```

The sample interval of `0` indicates that the data is streamed whenever an event occurs. Here, the event represents the updates to the cache state.

Related Commands:

- **show tech telemetry model-driven**
- **show running-config telemetry model-driven**
- **show telemetry producers trace** *producer name* **info**
- **show telemetry producers trace** *producer name* **err**

FQDN as the gRPC Tunnel Destination

Table 9: Feature History Table

Feature Name	Release Information	Description
FQDN as the gRPC Tunnel Destination	Release 7.10.1	You can now specify a fully qualified domain name (FQDN) as gRPC tunnel destination. FQDNs are easy to remember compared to numeric IP addresses and helps to resolve the domain names to IPv4 or IPv6 address and establish tunnel towards the destination. Model-driven telemetry and Event-driven telemetry are supported. The feature introduces these changes: **CLI:** • The keywords **address-family ipv4** and **address-family ipv6** are introduced in the **gRPC tunnel** command.

Starting Cisco IOS XR Software Release 7.10.1, you can configure a FQDN as the gRPC tunnel destination. A FQDN can be associated with multiple IP addresses of IPv4 and IPv6. Configuration address-family specifies the desired address-family for the returned addresses from DNS. This configuration is applicable only to FQDN as gRPC tunnel destination. The `get addr-info` returns the corresponding address(es) of the FQDN. If the **address-family ipv4** is configured, then only IPv4 address(es) are used to establish gRPC tunnel(s). Similarly, if the **address-family ipv6** is configured, only IPv6 address(es) are used to establish gRPC tunnel(s). If address-family configuration is not present, then IP addresses of both IPv4 and IPv6 address-family are considered for establishing gRPC tunnel(s).

For more information about gNMI dial-out via gRPC tunnel, see the Github repository.

Configure FQDN as the gRPC Tunnel Destination

Perform the following steps to configure FQDN as gRPC tunnel destination:

Procedure

Step 1 Configure FQDN as gRPC tunnel destination.

Example:

In this example, you set up FQDN as gRPC tunnel destination (IPv4):

```
Router#config
Router(config)#grpc
Router(config-grpc)#tunnel
Router(config-grpc-tunnel)#destination test.tunnel.dn port 59500
Router(config-grpc-tunnel-dest)#address-family ipv4
Router(config-grpc-tunnel-dest)#target xr
Router(config-grpc-tunnel-dest)#commit
```

Example:

In this example, you set up FQDN as gRPC tunnel destination (IPv6):

```
Router#config
Router(config)#grpc
Router(config-grpc)#tunnel
Router(config-grpc-tunnel)#destination test.tunnel.dn port 59500
Router(config-grpc-tunnel-dest)#address-family ipv6
Router(config-grpc-tunnel-dest)#target xr
Router(config-grpc-tunnel-dest)#commit
```

Step 2 Verify that the gRPC tunnel configuration is successful on the router.

Example:

The following example shows how to display the successful gRPC tunnel configuration (IPv4):

```
Router#show running-config grpc

grpc
tunnel
  destination test.tunnel.dn port 59500
   target xr
   address-family ipv4
  !
!
!
```

Example:

The following example shows how to display the successful gRPC tunnel configuration (IPv6):

```
Router#show running-config grpc

grpc
tunnel
  destination test.tunnel.dn port 59500
   target xr
   address-family ipv6
  !
!
!
```

Step 3 Verify the configured FQDN.

Example:

The following example shows how to display the successful FQDN configuration (IPv4 and IPv6):

```
Router#show host

Default domain is not set
Name/address lookup uses domain service
Name servers: 255.255.255.255
Host                       Flags    Age(hr) Type Address(es)
test.tunnel.dn            (perm, OK)   0      IP  192.168.0.2
test.tunnel.dn            (perm, OK)   0      IPV6  2001:DB8:A:B::1
```

```
Router#show running-config domain

domain ipv4 host test.tunnel.dn 192.168.0.2
domain ipv6 host test.tunnel.dn 2001:DB8:A:B::1
```

Step 4 Copy the public certificate for the collector to `/misc/config/grpc/<domain name>:<port>.pem` directory. The router uses this certificate to verify the tunnel server, and establish a dial-out session.

Step 5 Run the collector.

Step 6 View the status of tunnel sessions.

Example:

The following example shows how to display the status of the tunnel sessions:

```
Router#show grpc tunnel sessions

test.tunnel.dn:59500
Target:            xr
Status:            Connected
Listen address:    192.168.0.25:59500
Last connected:    2023-02-06 07:14:36

test.tunnel.dn:59500
Target:            xr
Status:            Connected
Listen address:    [2001:DB8:A:B::11]:59500
Last connected:    2023-02-06 07:15:23
```

Stream Telemetry Data about PBR Decapsulation Statistics

Table 10: Feature History Table

Feature Name	Release Information	Description
Stream Telemetry Data about PBR Decapsulation Statistics	Release 7.3.2	This feature streams telemetry data about header decapsulation statistics for traffic that uses the Policy-Based Routing (PBR) functionality to bypass a routing table lookup for egress. You use the `Cisco-IOS-XR-infra-policymgr-oper.yang` data model to capture the decapsulation data for Generic Routing Encapsulation (GRE) and Generic UDP Encapsulation (GUE) tunneling protocols. Decapsulation data helps you understand if all encapsulated packets are decapsulated and alerts you to issues if there is a mismatch in the number of packets.

You can stream telemetry data about PBR decapsulation statistics for GRE and GUE encapsulation protocols that deliver packets using IPv4 or IPv6. The encapsulated data has source and destination address that must match with the source and destination address in the classmap. Both encapsulation and decapsulation interfaces collect statistics periodically. The statistics can be displayed on demand using **show policy-map type pbr [vrf vrf-name] address-family ipv4/ipv6 statistics** command. For more information on PBR-based decapsulation, see *Interface and Hardware Component Configuration Guide for Cisco 8000 Series Routers*.

With this release, the decapsulation statistics can be displayed using `Cisco-IOS-XR-infra-policymgr-oper.yang` data model and telemetry data. You can stream telemetry data from the sensor path:

`Cisco-IOS-XR-infra-policymgr-oper:policy-manager/global/policy-map/policy-map-types/policy-map-type/vrf-table/vrf/afi-table/afi/stats`

The following steps show the PBR configuration and the decapsulation statistics that is streamed as telemetry data to the collector.

Procedure

Step 1 Check the running configuration to view the configured PBR per VRF.

Example:

```
Router#show running-config
Building configuration...
!! IOS XR Configuration 0.0.0
!!
vrf vrf1
 address-family ipv4 unicast
```

```
 !
 address-family ipv6 multicast
 !
!
netconf-yang agent
 ssh
!
!
class-map type traffic match-all cmap1
 match protocol gre
 match source-address ipv4 161.0.1.1 255.255.255.255
 match destination-address ipv4 161.2.1.1 255.255.255.255
 end-class-map
!
policy-map type pbr gre-policy
 class type traffic cmap1
  decapsulate gre
 !
 class type traffic class-default
 !
 end-policy-map
!
interface GigabitEthernet0/0/0/1
 vrf vrf1
 ipv4 address 2.2.2.2 255.255.255.0
 shutdown
!
vrf-policy
 vrf vrf1 address-family ipv4 policy type pbr input gre-policy
!
end
```

Step 2 View the output of the VRF statistics.

Example:

```
Router#show policy-map type pbr vrf vrf1 addr-family ipv4 statistics

VRF Name:       vrf1
Policy-Name:    gre-policy
Policy Type:    pbr
Addr Family:    IPv4

Class:     cmap1
     Classification statistics      (packets/bytes)
       Matched             :       13387587/1713611136
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :       13387587/1713611136

Class:     class-default
     Classification statistics      (packets/bytes)
       Matched             :              0/0
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :              0/0
```

After you have verified that the statistics are displayed correctly, stream telemetry data and check the streamed data at the collector. For more information about collectors, see *Operate on Telemetry Data for In-depth Analysis of the Network* section in the Monitor CPU Utilization Using Telemetry Data to Plan Network Infrastructure, on page 20 chapter.

```
ios.0/0/CPU0/ $ mdt_exec -s Cisco-IOS-XR-infra-policymgr-oper:policy-manager
/global/policy-map/policy-map-types/policy-map-type/vrf-table/vrf/afi-table/afi/stats -c 100
{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001","encoding_path":
"Cisco-IOS-XR-infra-policymgr-oper:policy-manager/global/policy-map/policy-map-types/policy-map-type
/vrf-table/vrf/afi-table/afi/stats","collection_id":"1","collection_start_time":"1601361558157",
```

```
"msg_timestamp":"1601361559179","data_json":[{"timestamp":"1601361559178","keys":[{"type":"ipv6"},
{"vrf-name":"vrf_gue_ipv4"},{"type":"ipv4"}],"content":{"pmap-name":"gre-policy","vrf-name":
"vrf1","appln-type":2,"addr-family":1,"rc":0,"plmgr-vrf-stats":[{"pmap-name":"gre-policy",
"cmap-stats-arr":[{"cmap-name":"cmap1","matched-bytes":"1713611136","matched-packets":"13387587",
"transmit-bytes":"1713611136","transmit-packets":"13387587"}]}]}}}],
"collection_end_time":"1601361559183"}
---------------------------------  snipped for brevity -------------------------------------
```


Stream Telemetry Data for LLDP Statistics

Table 11: Feature History Table

Feature Name	Release Information	Description
Stream Telemetry Data for LLDP Statistics	Release 24.2.11	You can now oversee and diagnose your network infrastructure in real time by periodically streaming the Link Layer Discovery Protocol (LLDP) information of a router through a gRPC Network Management Interface (gNMI) client. By continuously monitoring LLDP data from a switch or router, you gain immediate insights into network topology and the attributes of devices on the network, facilitating proactive management and troubleshooting.

Analyze Network Infrastructure Using LLDP Telemetry Data

Starting from Release 24.2.11, you can monitor Link Layer Discovery Protocol (LLDP) advertisement information, such as device identification, port identification, system capabilities, and management address using the gNMI `subscribe` operation.

Using the gNMI `subscribe` operation, you can subscribe to a stream of updates for specific paths within the device's data model. The following table lists the different subscription modes.

Table 12: Subscription Modes in a gNMI `Subscribe` Operation and Their Description

Subscription Mode	Description
Once	A one-time retrieval of the data at the specified paths.
Poll	Sets up a repeated send of the data at the specified paths at the input interval
Stream	Enables a continuous stream of updates from the server as changes occur. Within Stream mode, there are two types of subscriptions: • ◦ **Sample**: The server periodically sends the current state of the subscribed paths at a defined interval. • **On Change**: The server sends updates only when the values of the subscribed paths change.

gNMI Sensor Path to Stream LLDP Telemetry

You can stream telemetry data from the following gNMI sensor paths using **On Change** subscription mode:

- openconfig-lldp:lldp/state
- openconfig-lldp:lldp/interfaces/interface/state
- openconfig-lldp:lldp/interfaces/interface/neighbors/neighbor

Configure Telemetry for LLDP Statistics

Stream LLDP updates from a router or switch directly through the gNMI client.

Procedure

Step 1 Configure gNMI and LLDP

Example:

```
Router# config
Router(config)# grpc gnmi port 1024
Router(config)# lldp
Router(config)# commit
```

Step 2 Configure gRPC Request message on the gNMI Client

Example:

```
"openconfig-lldp:lldp":
  {
    "state":{}
  }
}
```

The gNMI client initiates a gRPC request message to get the latest LLDP information from the router.

Step 3 Verify gNMI response from the Router to the Client

Example:

```
"openconfig-lldp:lldp":
  {
    "state":
    {
      "enabled":true,
      "hello-timer":"30",
      "system-name":"ios",
      "system-description":"Cisco8k",
      "chassis-id":"abcd.abcd.abcd",
      "chassis-id-type":"MAC_ADDRESS",
      "counters":
      {
          "frame-in":"0",
          "frame-out":"0",
          "frame-error-in":"0",
          "frame-discard":"0",
          "tlv-discard":"0",
          "tlv-unknown":"0",
          "last-clear":"2024-06-05T16:09:36.388+05:30",
```

```
          "tlv-accepted":"0",
          "entries-aged-out":"0"
        }
      }
    }
  }
```


Stream Telemetry Data for ASIC Error Statistics

Table 13: Feature History Table

Feature Name	Release Information	Description
Stream Telemetry Data for ASIC Error Statistics	Release 24.2.11	You can now stream and monitor the telemetry data remotely on a gNMI interface, after subscribing to a sensor path. This data is gathered directly from the Network Processor Unit (NPU) driver at regular, predefined intervals for each block. This streaming enables real-time monitoring and analysis of router health and network performance, including error reporting and key metrics, allowing for rapid response to dynamic network conditions. Previously, you needed to log into the router to check the ASIC statistics.

Analyze Network Performance Using ASIC Telemetry Data

Starting from Cisco IOS XR Release 24.2.11, you can stream telemetry data of Application-Specific Integrated Circuit (ASIC) error statistics through a gRPC Network Management Interface (gNMI). The ASIC telemetry capabilities enable precise monitoring of individual ASICs within the router. To analyze network performance, you can stream telemetry data from various subsystems, including the ASIC Interface, Queueing, Lookup, Host Interface, and Fabric Interface, and examine their error counters.

The telemetry data is streamed for a particular ASIC within a router by using its comprehensive packet, drop, and error counters. Telemetry data is collected for the following critical errors through gNMI.

- All reset action interrupts
- Link Errors
- Memory corruption interrupts multi bit error (MBE), single bit error (SBE), and parity (Error detection).
- Packet loss interrupts on threshold reached

The error counters allow for a granular assessment of the router's operational health, with the data which is collected directly from the Network Processing Unit (NPU) driver at a specified frequency for each block.

ASIC Error Types Tracked Using gNMI

Telemetry data is collected for the following types of critical errors through gNMI:

Error Type	Details
Reset action interrupts	These interrupts includes the following types of reset actions: • Hard Reset—It doesn't stop any processes but reinitializes the underlying ASIC. • PON Reset—The power is switched OFF and switched ON. • Replace Device—The board gets shut down.
Link Errors	These are malfunctions or disruptions in the data connections.
Memory corruption interrupts (Multibit error (MBE), Single-bit error (SBE), Parity (Error detection))	These result in unintended alterations to the data in the chip's memory, causing unexpected behavior or failures.
Packet loss interrupts on threshold reached	These interrupts trigger an alert when there's a potential drop of packets. You can configure the threshold duration to three minutes or five minutes. When there's a potential drop of packets in this configuration duration continuously, the syslog is updated and notified to the user.

gNMI Sensor Path to Stream ASIC Error Telemetry

You can stream telemetry data from the gNMI sensor path: `openconfig-platform:components/component/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors`.

Configure Telemetry for ASIC Error Statistics

Procedure

Step 1 Verify your gRPC configuration by running the **show running-config grpc** command.

Example:

```
Router# show running-config grpc
Wed May 15 01:10:57.595 UTC
grpc
 port 57400
 no-tls
 max-streams 128
 max-streams-per-user 128
 address-family dual
 service-layer
 !
 local-connection
 max-request-total 256
```

```
 max-request-per-user 32
!
```

Step 2 View the output of the ASIC interrupts statistics, stream the telemetry data for the following ASIC Errors:

- View Reset Action Interrupts, on page 58
- View Link Errors, on page 59
- View Memory Corruption Interrupts, on page 61
- View Packet Loss Interrupts, on page 62

View Reset Action Interrupts

To view Reset action interrupts on the router, use the **show asic-errors all detail location** command.

```
Router# show asic-errors all detail location 0/0/cpu0
Thu May 16 09:28:57.001 UTC
************************************************************
*                          0_0_CPU0                        *
************************************************************
************************************************************
*                   NPU ASIC Error Summary                 *
************************************************************
************************************************************
*                       Instance : 0                       *
************************************************************
************************************************************
*                        Reset Errors                      *
************************************************************
8000, 88-LC0-36FH, 0/0/CPU0, npu[0]
Name              : hard-reset
Block ID          : 0x6
Addr              : 0x100
Leaf ID           : 0xc02002
Thresh/period(s): 5/day
Error count       : 1
Last clearing     : Thu May 16 09:27:42 2024
Last N errors     : 1
--------------------------------------------------------------
First N errors.
@Time, Error-Data
------------------------------------------
May 16 09:27:42.942652
 Error description: Id:169 Bit:0x2 Action:hard-reset OTHER: NTS-09:27:42.942729
STS-09:27:42.942652
--------------------------------------------------------------
```

gNMI Output for ASIC Reset Action Errors

Following is the corresponding gNMI output for the `Reset Action` ASIC errors after subscribing to the reset action interrupt:


```
{
  "source": "1.1.111.3:57400",
  "subscription-name": "default-1715847417",
  "timestamp": 1715851665318438321,
  "time": "2024-05-16T02:27:45.318438321-07:00",
  "prefix": "openconfig-platform:",
```

```
  "updates": [
    {
      "Path":
"components/component[name=0/0/CPU0-NPU0]/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors",

      "values": {

"components/component/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors":
 {
          "queueing-block": {
            "queueing-block-error": {
              "name": "dics.general_interrupt_register.dics2mmu_fifo_overflow",
              "state": {
                "action": [
                  "NPU_RESET"
                ],
                "count": "1",
                "level": "MAJOR",
                "name": "dics.general_interrupt_register.dics2mmu_fifo_overflow",
                "threshold": "0"
              }
            }
          }
        }
      }
    }
  ]
}
```

View Link Errors

To view link error statistics, use the **show asic-errors all detail location** command.

```
Router# show asic-errors all detail location 0/rp0/cpu0
Thu May 16 14:28:38.474 UTC
************************************************************
*                        0_RP0_CPU0                        *
************************************************************
************************************************************
*                  NPU ASIC Error Summary                   *
************************************************************
…
…
…
************************************************************
*                      Instance : 6                        *
************************************************************
************************************************************
*                       Reset Errors                       *
************************************************************
************************************************************
*                    Single Bit Errors                     *
************************************************************
************************************************************
*                   Multiple Bit Errors                    *
************************************************************
************************************************************
*                      Parity Errors                       *
************************************************************
************************************************************
*                    Unexpected Errors                     *
************************************************************
************************************************************
*                       Link Errors                        *
```

```
************************************************************
8000, 8804-FC0, 0/FC6, npu[6]
Name            : slice[5].ifg[0].mac_pool8[1].rx_link_status_down.rx_link_status_down4
Block ID        : 0x21c
Addr            : 0x100
Leaf ID         : 0x43802004
Thresh/period(s): 100/day
Error count     : 2
Last clearing   : Thu May 16 14:27:32 2024
Last N errors   : 2
--------------------------------------------------------------
First N errors.
@Time, Error-Data
------------------------------------------
May 16 14:27:32.282300
 Error description: Id:49 Bit:0x4 Action:port-reset LINK_BAD: (Slc/Ifg/FstSer):(5/0/12)
NTS-14:27:32.282423 STS-14:27:32.282300
May 16 14:27:32.342088
 Error description: Id:50 Bit:0x4 Action:none LINK_DN: (Slc/Ifg/FstSer):(5/0/12) RxLnDnSt:1
 RxRmtLnStDn:0 PcsLnStDn:1 PcsAlnStDn:1 HiBer:0 HiSerIntr:0 SigLos:0 1 1 0 0 0 0 0
FifoOverflow:0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  Link:220
NTS-14:27:32.342427 STS-14:27:32.342088 TEMP_SENSOR:44.0C
--------------------------------------------------------------
```

gNMI Output for ASIC Link Errors

Following is the corresponding gNMI output for the `Link errorr` ASIC errors after subscribing to the Link error interrupt:


```
{
  "source": "1.1.111.3:57400",
  "subscription-name": "default-1715865468",
  "timestamp": 1715869652342383198,
  "time": "2024-05-16T07:27:32.342383198-07:00",
  "prefix": "openconfig-platform:",
  "updates": [
    {
      "Path":
"components/component[name=0/FC6-NPU6]/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors",

      "values": {

"components/component/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors":
 {
          "interface-block": {
            "interface-block-error": {
            "name": "slice[5].ifg[0].mac_pool8[1].rx_link_status_down.rx_link_status_down4",

              "state": {
                "action": [
                  "LOG"
                ],
                "count": "1",
                "level": "MINOR",
                "name":
"slice[5].ifg[0].mac_pool8[1].rx_link_status_down.rx_link_status_down4",
                "threshold": "0"
              }
            }
          }
        }
      }
    }
```

```
  ]
}
```

View Memory Corruption Interrupts

To view memory corruption statistics, use the **show asic-errors all detail location** command.

```
Router# show asic-errors all detail location 0/0/cpu0
Thu May 16 20:33:49.135 UTC
************************************************************
*                         0_0_CPU0                         *
************************************************************
************************************************************
*                  NPU ASIC Error Summary                  *
************************************************************
************************************************************
*                      Instance : 0                        *
************************************************************
************************************************************
*                       Reset Errors                       *
************************************************************
************************************************************
*                    Single Bit Errors                     *
************************************************************
8000, 88-LC0-36FH, 0/0/CPU0, npu[0]
Name            : slice[0].ifg[0].sch.vsc_token_bucket_cfg
Block ID        : 0xc1
Addr            : 0x400000
Leaf ID         : 0x4000800
Thresh/period(s): 100/day
Error count     : 1
Last clearing   : Thu May 16 20:32:11 2024
Last N errors   : 1
--------------------------------------------------------------
First N errors.
@Time, Error-Data
-------------------------------------------
May 16 20:32:11.966346
 Error description: Id:124 Bit:0x0 Action:none MEM_PROTECT: Err:0x0 Inst_Addr:0x400000
Entry:0x0 NTS-20:32:11.966447 STS-20:32:11.966346
--------------------------------------------------------------
```

gNMI Output for ASIC Memory Corruption Interrupt Errors

Following is the corresponding gNMI output for the `Memory Corruption` ASIC errors after subscribing to the memory corruption interrupt:


```
{
  "source": "1.1.111.3:57400",
  "subscription-name": "default-1715887309",
  "timestamp": 1715891533779872124,
  "time": "2024-05-16T13:32:13.779872124-07:00",
  "prefix": "openconfig-platform:",
  "updates": [
    {
      "Path":
"components/component[name=0/0/CPU0-NPU0]/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors",

      "values": {

"components/component/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors":
 {
          "queueing-block": {
```

```
              "queueing-block-error": {
                "name": "slice[0].ifg[0].sch.vsc_token_bucket_cfg_ecc1b",
                "state": {
                  "action": [
                    "LOG"
                  ],
                  "count": "1",
                  "level": "MINOR",
                  "name": "slice[0].ifg[0].sch.vsc_token_bucket_cfg_ecc1b",
                  "threshold": "0"
                }
              }
            }
          }
        }
      }
    ]
  }
```


View Packet Loss Interrupts

Prerequisites

Configure the packet loss beyond threshold level using the **hw-module profile packet-loss-alert** commad.

Monitor Packet Loss Interrupts

To view packet loss statistics, use the **show asic-errors npu 0 generic location** command.

```
Router# sh asic-errors npu 0 generic location 0/0/CPU0
Fri May 17 04:15:04.569 UTC
************************************************************
*                          0_0_CPU0                        *
************************************************************
************************************************************
*                       Generic Errors                     *
************************************************************
8000, 88-LC0-36FH-M, 0/0/CPU0, npu[0]
Name            : slice[0].tx.pdr.general_interrupt_register.mc_flb_to_uc_oq
Block ID        : 0x1f
Addr            : 0x100
Leaf ID         : 0x3e02000
Thresh/period(s): 1/day
Error count     : 36
Last clearing   : Fri May 17 03:52:12 2024
Last N errors   : 36
--------------------------------------------------------------
First N errors.
@Time, Error-Data
------------------------------------------
May 17 03:52:12.018091
        Error description: Id:198 Bit:0x0 Action:packet-loss OTHER: NTS-03:52:12.018236
STS-03:52:12.018091
May 17 03:52:17.023359
        Error description: Id:200 Bit:0x0 Action:packet-loss OTHER: NTS-03:52:17.023462
STS-03:52:17.023359
Last N errors.
@Time, Error-Data
------------------------------------------
May 17 03:54:17.123480
        Error description: Id:248 Bit:0x0 Action:packet-loss OTHER: NTS-03:54:17.123566
STS-03:54:17.123480
```

```
May 17 03:54:22.128472
        Error description: Id:250 Bit:0x0 Action:packet-loss OTHER: NTS-03:54:22.128562
STS-03:54:22.128472
        Error description: Id:268 Bit:0x0 Action:packet-loss OTHER: NTS-03:55:07.164378
STS-03:55:07.164275
--------------------------------------------------------------
```

gNMI Output for ASIC Packet Loss Interrupt Errors

Following is the corresponding gNMI output for the `Packet Loss` ASIC errors after subscribing to the packet loss interrupt:


```
{
  "source": "6.4.91.115:57400",
  "subscription-name": "default-1715917870",
  "timestamp": 1715918114950928813,
  "time": "2024-05-16T20:55:14.950928813-07:00",
  "prefix": "openconfig-platform:",
  "updates": [
    {
      "Path":
"components/component[name=0/0/CPU0-NPU0]/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors",

      "values": {

"components/component/integrated-circuit/openconfig-platform-pipeline-counters:pipeline-counters/errors":
 {
          "queueing-block": {
            "queueing-block-error": {
              "name": "slice[0].tx.pdr.general_interrupt_register.mc_flb_to_uc_oq",
              "state": {
                "action": [
                  "LOG"
                ],
                "count": "1",
                "level": "INFORMATIONAL",
                "name": "slice[0].tx.pdr.general_interrupt_register.mc_flb_to_uc_oq",
                "threshold": "0"
              }
            }
          }
        }
      }
    }
  ]
}
```


Trouble Shoot ASIC Errors

Error Category	Specific Error Message	Action
All reset actions	NA	After performing Power On Reset (PON—reload) reset and hard-reset, if the error persists, collect **show tech fabric link-include** statistics and reach out to CISCO TAC.
RMA (Return Merchandise Authorization)	NA	Collect **show tech fabric link-include** statistics and reach out to CISCO TAC.

Error Category	Specific Error Message	Action
Link	NA	Collect **show tech fabric link-include** statistics and reach out to CISCO TAC.
Memory Corruption	SBE	On the 100^{th} error, the board is reloaded and SBE errors are correctable.
	MBE/PARITY	Router gets reloaded on hard-reset for the fifth error. If the error persists, collect **show tech fabric** statistics and reach out to CISCO TAC.
Packet Loss Beyond Threshold	NA	Isolate the board, collect the following statistics, and reach out to CISCO TAC: • **show tech fabric link-include** • **show tech qos** • **show tech ofa**

Stream telemetry for IPv4 and IPv6 data on network interfaces

Streaming telemetry for IPv4 and IPv6 data on network interfaces is a method of continuously collecting and transmitting real-time data using standardized OpenConfig data models and gNMI sensor paths. This approach:

- enables real-time monitoring and reporting of IPv4 and IPv6 operational states and configuration changes, and
- improves network management with standardized data models for seamless multi-vendor compatibility,

Table 14: Feature History Table

Feature Name	Release Information	Description
Stream telemetry for IPv4 and IPv6 data on network interfaces	Release 25.2.1	Introduced in this release on: Fixed Systems (8200 [ASIC: Q200, P100], 8700 [ASIC: P100, K100], 8010 [ASIC: A100]); Centralized Systems (8600 [ASIC:Q200]); Modular Systems (8800 [LC ASIC: Q200, P100]) You can now enhance network reliability and resource optimization by monitoring IPv4 and IPv6 performance and operational states across platforms. This ensures consistent management, proactive troubleshooting, and optimization in multi-vendor environments using openconfig-if-ip.yang data models and telemetry-enabled sensor paths.

Streaming telemetry data for openconfig data model through gNMI supports monitoring of various data points, include:

- operational status,
- configuration changes, and
- performance metrics.

gNMI sensor paths to stream IPv4 and IPv6 telemetry data

You can stream telemetry data from these gNMI sensor paths using On Change subscription mode or at a cadence of 30 seconds or higher (with a scale of 2000 interfaces). For more details on gNMI subscription, see the GitHub repository.

- openconfig-interfaces/interface/state

IPv4 sub-interface level:

- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv4/state
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv4/addresses
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv4/neighbor
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv4/proxy-arp

IPv6 sub-interface level:

- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/state
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/addresses
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/router-advertisement
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/openconfig-if-ip-ext:autoconf
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/neighbor

Verify telemetry data for IPv4 and IPv6 on network interfaces

You can verify the telemetry data for IPv4 and IPv6 data on network interfaces.

Procedure

Verify the output of the interface IP statistics.

Example:

This example shows IPv4 state information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.1:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv4/state'
{
    "source": "192.168.2.1:17933",
    "timestamp": 1746203252773061780,
    "time": "2025-05-02T12:27:32.77306178-04:00",
```

```
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv4/state",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv4/state": {
            "counters": {
              "in-multicast-octets": "0",
              "in-multicast-pkts": "0",
              "in-octets": "0",
              "in-pkts": "0",
              "out-multicast-octets": "0",
              "out-multicast-pkts": "0",
              "out-octets": "0",
              "out-pkts": "0"
            },
            "dhcp-client": false,
            "mtu": 1500
          }
        }
      }
    ]
  }
]
```


Example:

This example shows IPv4 address information.

```
auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.2:17933 --username xxxxx --password xxxxxxxx
--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv4/addresses'
```


```
[
  {
    "source": "192.168.2.2:17933",
    "timestamp": 1746203286230765971,
    "time": "2025-05-02T12:28:06.230765971-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv4/addresses",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv4/addresses": {
            "address": [
              {
                "config": {
                  "ip": "192.168.10.1",
                  "prefix-length": 24,
                  "type": "PRIMARY"
                },
                "ip": "192.168.20.1",
                "state": {
                  "ip": "192.168.20.1",
                  "origin": "STATIC",
                  "prefix-length": 24,
                  "type": "PRIMARY"
                }
              }
            ]
          }
        }
      }
```

```
    ]
  }
]
```

Example:

This example shows IPv4 neighbor information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.3:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv4/neighbors'
[
  {
    "source": "192.168.2.3:17933 ",
    "timestamp": 1746203327097683095,
    "time": "2025-05-02T12:28:47.097683095-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv4/neighbors",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv4/neighbors": {
            "neighbor": [
              {
                "ip": "192.168.20.1",
                "state": {
                  "ip": "192.168.20.1",
                  "link-layer-address": "78:bf:38:b6:66:08",
                  "origin": "OTHER"
                }
              },
              {
                "ip": "192.168.20.2",
                "state": {
                  "ip": "192.168.20.2",
                  "link-layer-address": "00:00:00:00:00:00",
                  "origin": "OTHER"
                }
              }
            ]
          }
        }
      }
    ]
  }
]
```

Example:

This example shows IPv4 proxy-arp information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.4:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv4/proxy-arp'
[
  {
    "source": "192.168.2.4:17933",
    "timestamp": 1746203562540052546,
    "time": "2025-05-02T12:32:42.540052546-04:00",
    "updates": [
      {
        "Path":
```

```
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv4/proxy-arp",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv4/proxy-arp": {
            "config": {
              "mode": "ALL"
            },
            "state": {
              "mode": "ALL"
            }
          }
        }
      }
    ]
  }
]
```

Example:

IPv6 state

This example shows IPv6 state information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.5:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/state'
[
  {
    "source": "192.168.2.5:17933",
    "timestamp": 1746202812260230182,
    "time": "2025-05-02T12:20:12.260230182-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv6/state",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv6/state": {
            "counters": {
              "in-multicast-octets": "0",
              "in-multicast-pkts": "0",
              "in-octets": "0",
              "in-pkts": "0",
              "out-multicast-octets": "0",
              "out-multicast-pkts": "0",
              "out-octets": "0",
              "out-pkts": "0"
            },
            "dup-addr-detect-transmits": 5,
            "enabled": true,
            "mtu": 1500
          }
        }
      }
    ]
  }
]
```

Example:

This example shows IPv6 address information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.6:17933 --username xxxxx --password xxxxxxxx
```

```
--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/addresses'
```

```
[
  {
    "source": "192.168.2.6:17933",
    "timestamp": 1746202852330541300,
    "time": "2025-05-02T12:20:52.3305413-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv6/addresses",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv6/addresses": {
            "address": [
              {
                "config": {
                  "ip": "10:10:3::1",
                  "prefix-length": 119,
                  "type": "GLOBAL_UNICAST"
                },
                "ip": "10:10:3::1",
                "state": {
                  "ip": "10:10:3::1",
                  "origin": "STATIC",
                  "prefix-length": 119,
                  "status": "PREFERRED",
                  "type": "GLOBAL_UNICAST"
                }
              },
              {
                "ip": "fe80::7abf:38ff:feb6:6608",
                "state": {
                  "ip": "fe80::7abf:38ff:feb6:6608",
                  "origin": "STATIC",
                  "prefix-length": 128,
                  "status": "PREFERRED",
                  "type": "LINK_LOCAL_UNICAST"
                }
              }
            ]
          }
        }
      }
    ]
  }
]
```


Example:

This example shows IPv6 neighbor information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.7:17933 --username xxxxx --password xxxxxxxx
 --
skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/neighbor'
```

```
[
  {
    "source": "192.168.2.7:17933",
    "timestamp": 1746202898868407604,
    "time": "2025-05-02T12:21:38.868407604-04:00",
    "updates": [
      {
        "Path":
```

```
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]",

        "values": {
          "interfaces/interface/subinterfaces/subinterface": null
        }
      }
    ]
  }
]
```


Example:

This example shows IPv6 state duplicate addres transmits information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.8:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/state/dup-addr-detect-transmits'
```


```
[
  {
    "source": "192.168.2.8:17933",
    "timestamp": 1746202980834877546,
    "time": "2025-05-02T12:23:00.834877546-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv6/state/dup-addr-detect-transmits",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv6/state/dup-addr-detect-transmits": 5

        }
      }
    ]
  }
]
```


Example:

This example shows IPv6 router advertisement information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.9:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/router-advertisement/'
```


```
[
  {
    "source": "192.168.2.9:17933",
    "timestamp": 1746202744369280518,
    "time": "2025-05-02T12:19:04.369280518-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv6/router-advertisement",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv6/router-advertisement": {
            "config": {
              "enable": true,
              "interval": 4,
              "lifetime": 9000,
              "managed": false,
              "other-config": true,
              "suppress": true
            },
```

```
              "prefixes": {
                "prefix": [
                  {
                    "config": {
                      "disable-autoconfiguration": true,
                      "enable-onlink": true,
                      "preferred-lifetime": 5000,
                      "prefix": "300:0:2::/124",
                      "valid-lifetime": 6000
                    },
                    "prefix": "300:0:2::/124",
                    "state": {
                      "disable-autoconfiguration": true,
                      "enable-onlink": true,
                      "preferred-lifetime": 5000,
                      "prefix": "300:0:2::/124",
                      "valid-lifetime": 6000
                    }
                  }
                ]
              },
              "state": {
                "enable": true,
                "interval": 4,
                "lifetime": 9000,
                "managed": false,
                "other-config": true,
                "suppress": true
              }
            }
          }
        }
      ]
    }
]
```


Timestamp in nano seconds

From Release 25.2.1, the telemetry messages for all sensor paths are populated with the `timestamp_nano` attribute. This is the time at which the data is collected from the underlying source, or the time that the message is generated, if provided by the underlying source. This is the number of nanoseconds since the Unix Epoch. For reference telemetry messages, see Github.

The primary benefit is the improved timestamp accuracy, which is now in nanoseconds rather than the milliseconds available earlier. Additionally, XR dial-in and dial-out telemetry includes the `timestamp_nano` field in the telemetry messages, ensuring more precise time tracking.