Telemetry Configuration Guide for Cisco NCS 5500 Series Routers, IOS XR Release 7.10.x

First Published: 2023-06-30

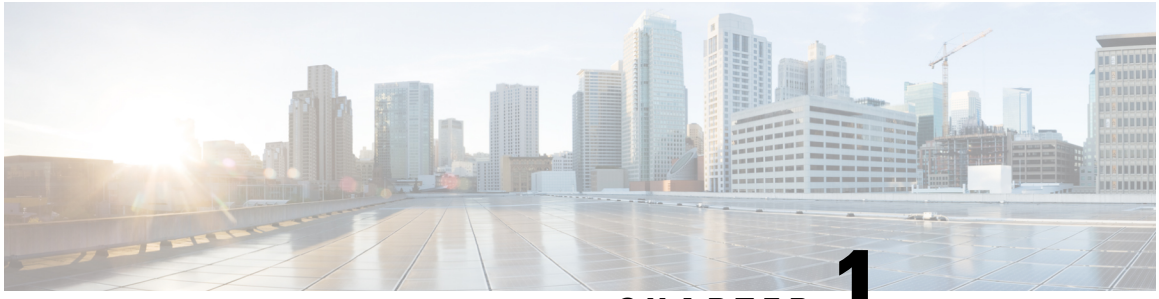
Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

- To receive timely, relevant information from Cisco, sign up at Cisco Profile Manager.
- To get the business impact you're looking for with the technologies that matter, visit Cisco Services.
- To submit a service request, visit Cisco Support.
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit Cisco Marketplace.
- To obtain general networking, training, and certification titles, visit Cisco Press.
- To find warranty information for a specific product or product family, access Cisco Warranty Finder.

Cisco Bug Search Tool

Cisco Bug Search Tool (BST) is a web-based tool that acts as a gateway to the Cisco bug tracking system that maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. BST provides you with detailed defect information about your products and software.

CONTENTS



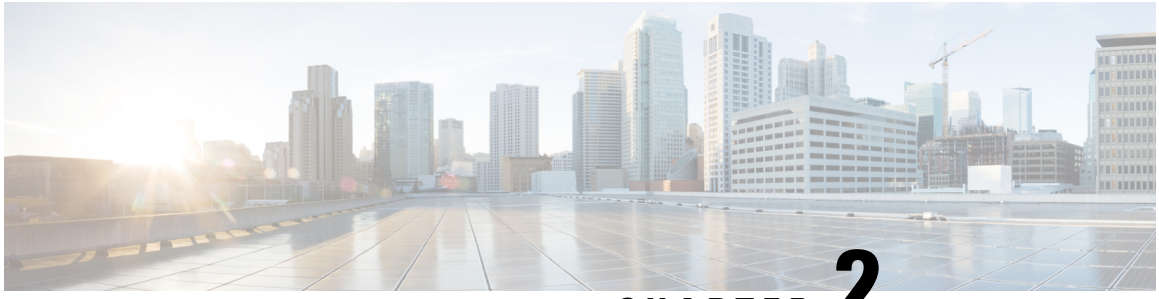
CHAPTER 1

New and Changed Feature Information

This section lists all the new and changed features for the *Telemetry Configuration Guide for Cisco NCS 5500 Series Routers*.

New and Changed Telemetry Features

Feature	Description	Changed in Release	Where Documented
None	No new features introduced	Not applicable	Not applicable



CHAPTER 2

YANG Data Models for Telemetry Features

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

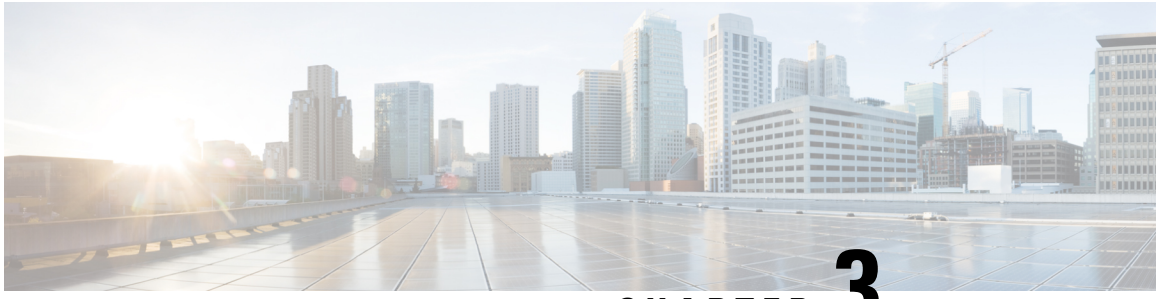
Using YANG Data Models

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?

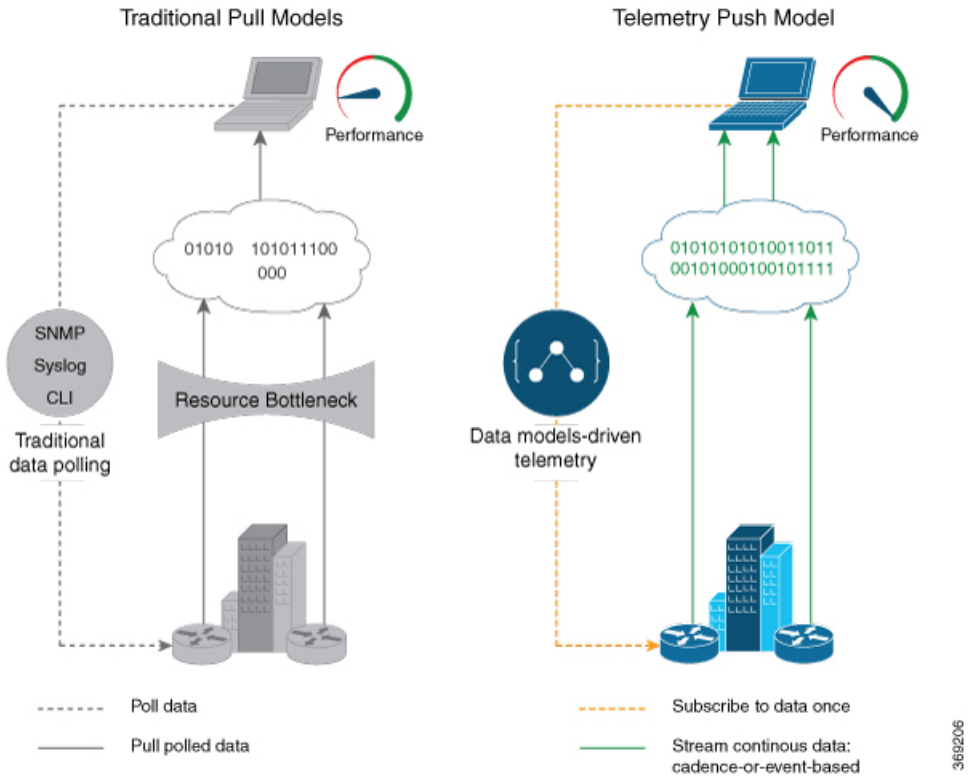
These traditional methods use a *pull* model to request information at regular intervals. The data that you collect may help you to efficiently monitor your network of a manageable size. However, as your network grows in complexity and scale, the data that you poll may be insufficient for efficient and 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:

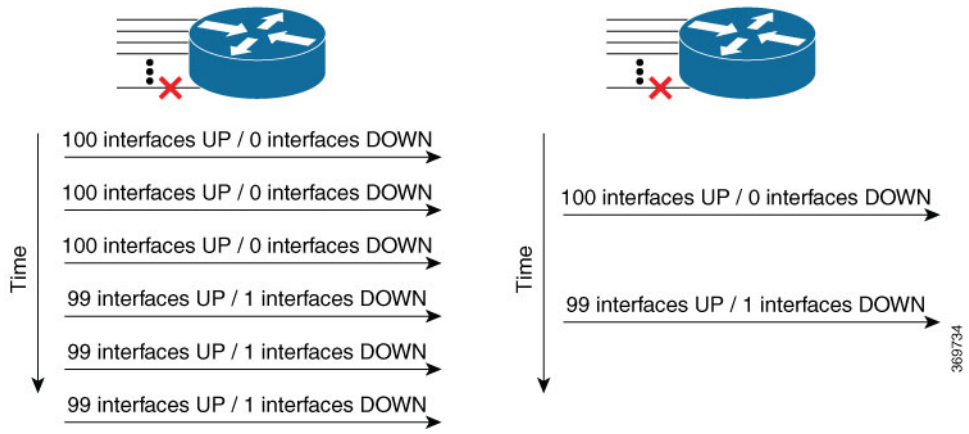
- **Managing network remotely:** The primary benefit of telemetry is the ability it offers you as an end user to monitor the state of a network element remotely. After the network is deployed, you cannot be physically present at the network site to find out what works, and what is cumbersome. With telemetry, 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

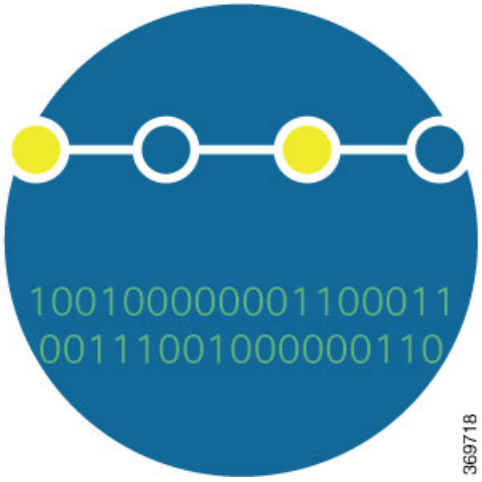


Cadence-driven Telemetry

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

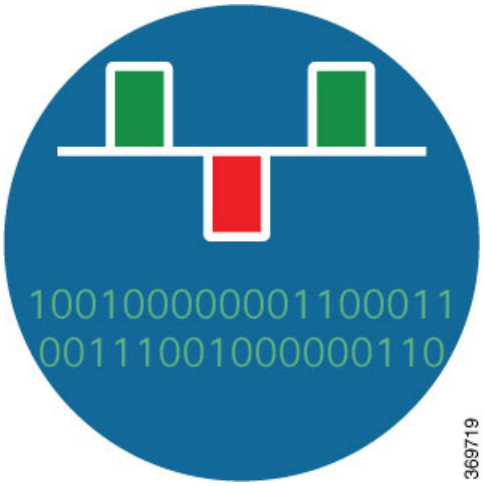


Event-driven Telemetry

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 1: 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/ports/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

Note Only the following Sysadmin data models are supported for streaming telemetry:

- Cisco-IOS-XR-sysadmin-controllers-ncs5500
- Cisco-IOS-XR-sysadmin-entity-mib
- Cisco-IOS-XR-sysadmin-entity-sensor-mib
- Cisco-IOS-XR-sysadmin-envmon-ui
- Cisco-IOS-XR-sysadmin-asic-errors-ael
- Cisco-IOS-XR-sysadmin-show-media

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

The following table shows few examples of sensor paths:

Table 2: 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`

Feature	Sensor Path

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`

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. This is not ideally suitable for a busy network. 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.

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 For Dial-in

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.

The TLS certificate must be copied at the `/misc/config/grpc/dialout/` path. If only the `protocol grpc` command is configured, by default, TLS is enabled and the hostname defaults to the IP address of the destination. In addition, in the certificate, configure the Common Name (CN) as `protocol grpc tls-hostname <>`.

The following 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
```

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.

Filter Telemetry Data Using Regex Key

Table 3: 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.

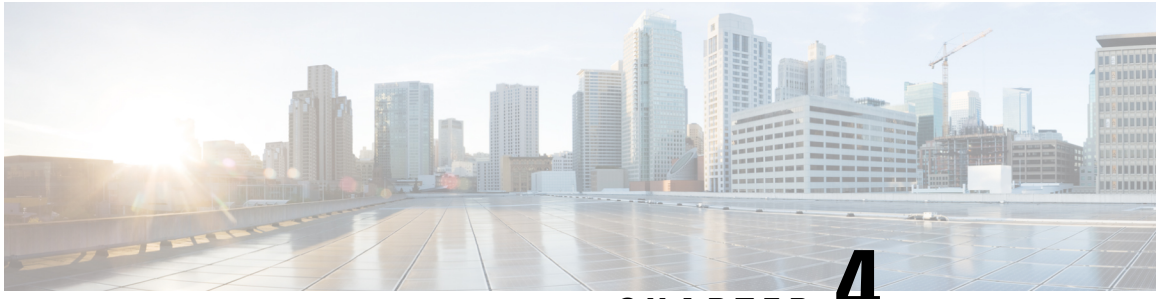
To explore the dial-in mode, and to create a dial-in session, see Establish a Model-Driven Telemetry Session from a Collector to a Router, on page 29.

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.

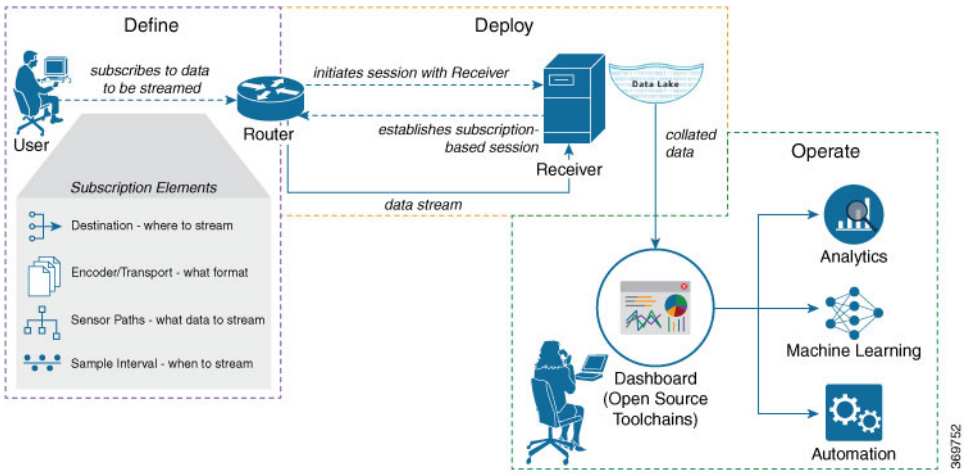
Telemetry works on a Subscription model where you subscribe to the data of interest in the form of Sensor Path. The sensor paths describes OpenConfig data models or native Cisco data models. You can access the OpenConfig and Native data models for telemetry from Github, a software development platform 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 theDial-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.

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
```

where -

- `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
```

where -

- `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 Telemetry 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.

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.

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 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 analysed 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.

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
```

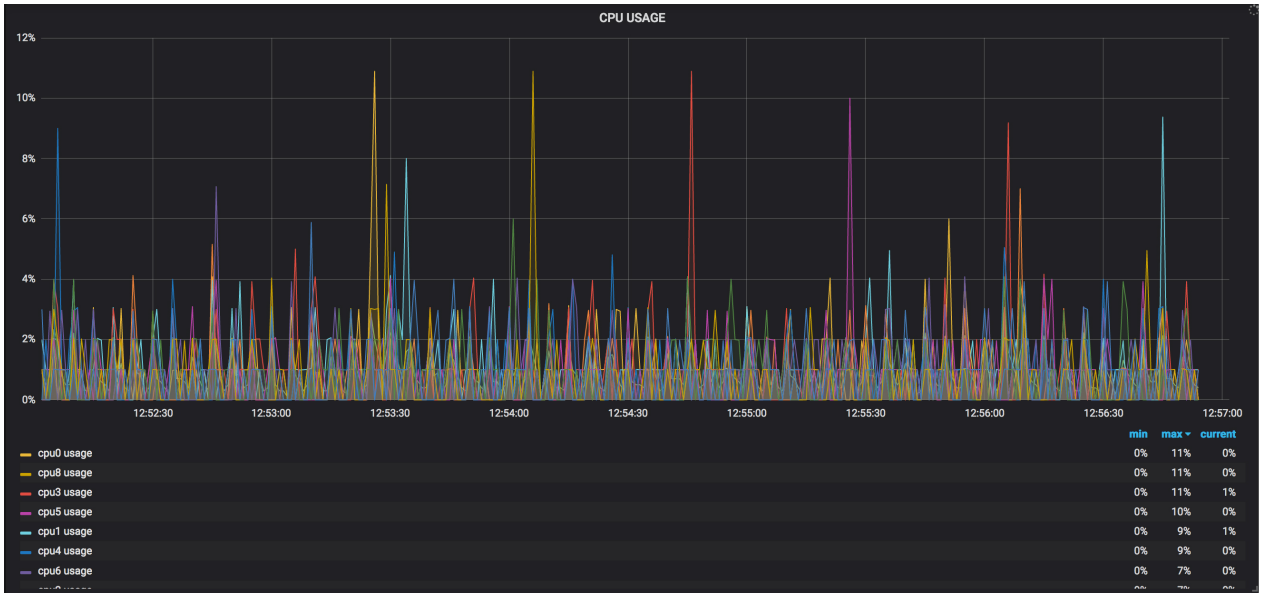
Step 3 Use Grafana to create a dashboard and visualize data about CPU usage.

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

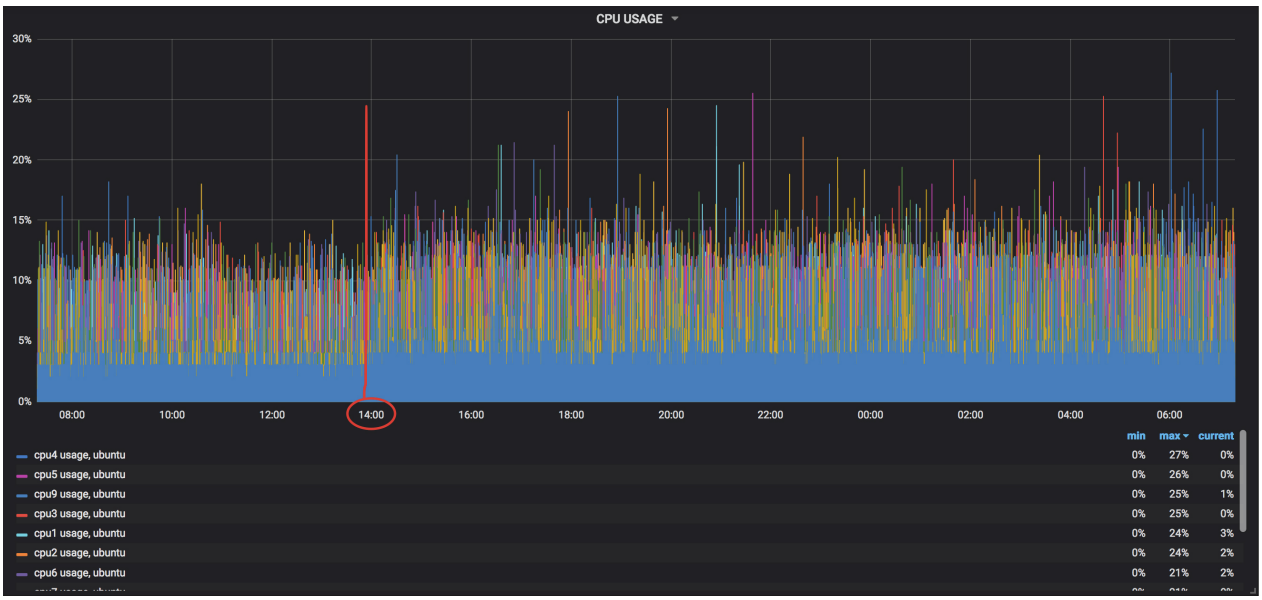


369756

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 midpoint value at 15 percent.

Figure 7: CPU Usage Graph with Two Routers

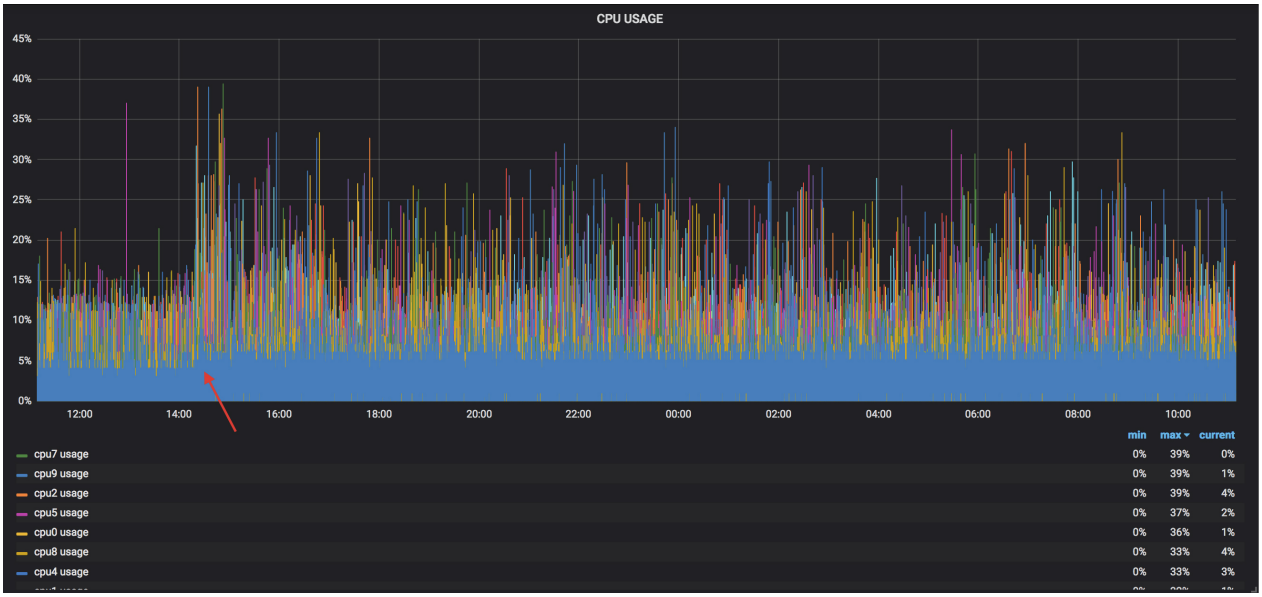


369757

Five routers

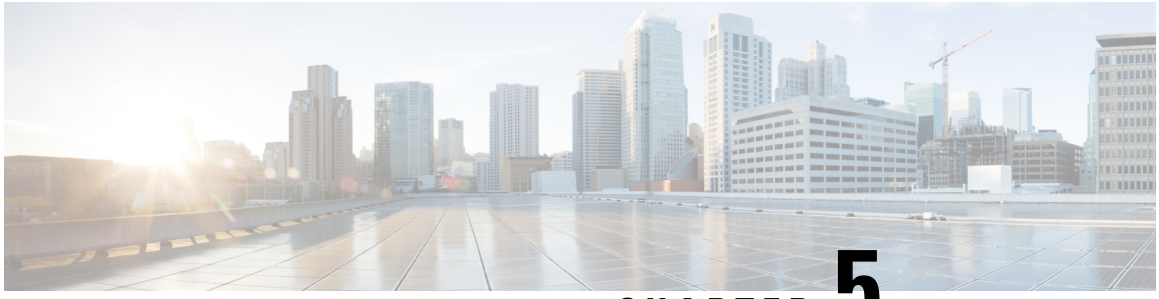
With five routers, the spikes peak upto approximtely 40 percent with midpoint in the range of 22 to 25 percent.

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.

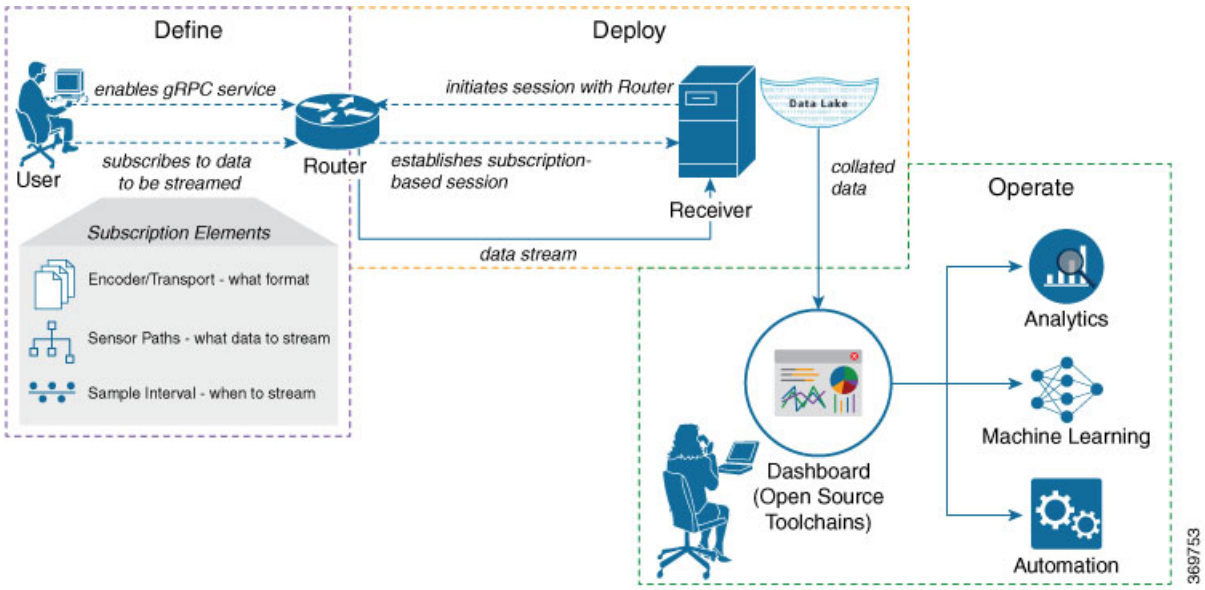
Telemetry works on a Subscription model where you subscribe to the data of interest in the form of Sensor Path. The sensor paths describes OpenConfig data models or native Cisco data models. You can access the OpenConfig and Native data models for telemetry from Github, a software development platform 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.
- Configure a third-party application (TPA) source address. This address sets a source hint for Linux applications, so that the traffic originating from the applications can be associated to any reachable IP on the router.

```
Router(config)#tpa
Router(config)#address-family ipv4
Router(config-af)#update-source dataports TenGigE0/6/0/0/1
```

  Use the following configuration if VRF is used:

```
Router(config)#tpa
Router(config)#vrf <vrf-name>
Router(config-vrf)#address-family ipv4
Router(config-vrf)#update-source dataports TenGigE0/6/0/0/1
```

  A default route is automatically gained in the Linux shell.

The following example shows the output of the gRPC configuration with TLS enabled on the router.

```
Router#show grpc
Address family : ipv4
Port : 57350
DSCP : Default
VRF : global-vrf
TLS : enabled
TLS mutual : disabled
Trustpoint : none
Maximum requests : 128
Maximum requests per user : 10
Maximum streams : 32
Maximum streams per user : 32
TLS cipher suites
Default : none
Enable : none
Disable : none
Operational enable : ecdhe-rsa-chacha20-poly1305
: ecdhe-ecdsa-chacha20-poly1305
: ecdhe-rsa-aes128-gcm-sha256
```

```
          : ecdhe-ecdsa-aes128-gcm-sha256
          : ecdhe-rsa-aes128-sha
          Operational disable : none
```

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.

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 Telemetry or Event-driven Telemetry. 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.

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

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
```

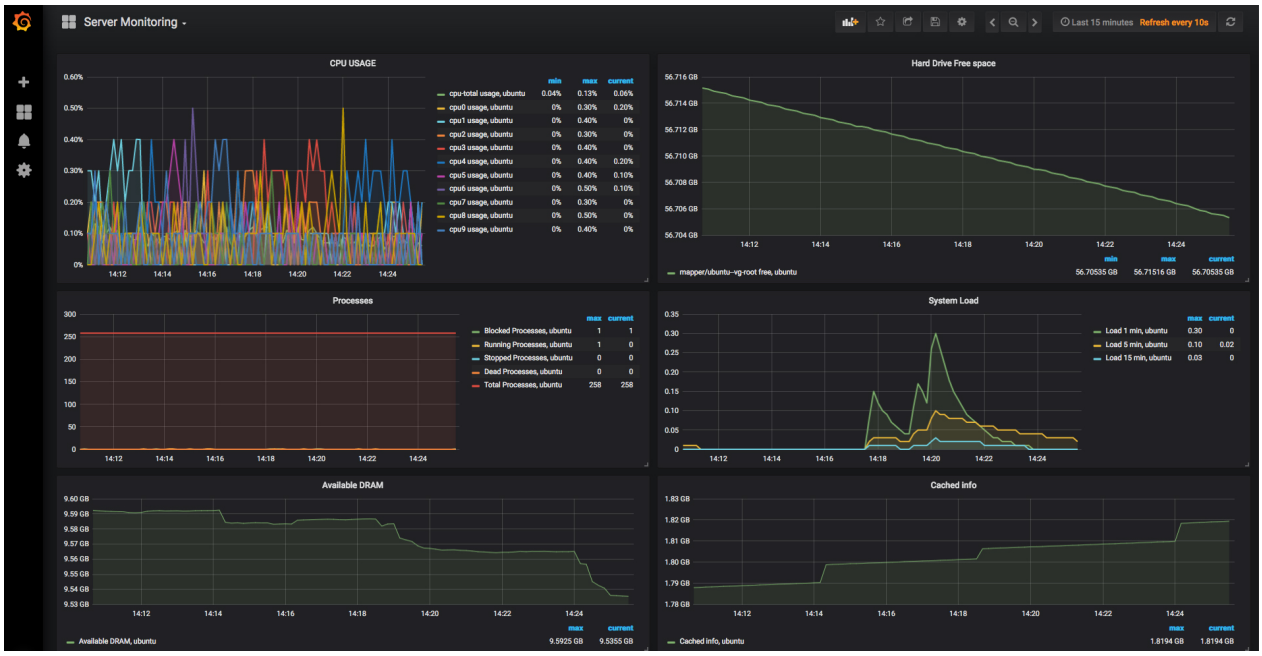
The streamed telemetry data is stored in InfluxDB.

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

Figure 10: Visual Analysis of Network Health using Telemetry Data

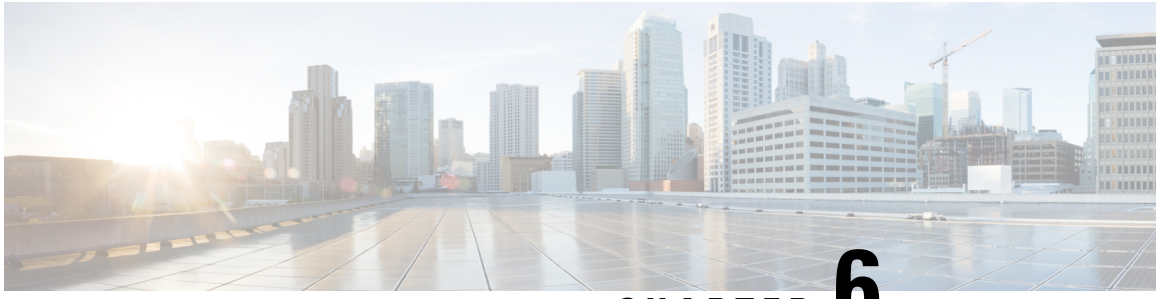
369765

Figure 11: Visual Analysis of System Monitoring using Telemetry Data



369766

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

Build Intelligence on the Router Using AI-Driven Telemetry

Table 4: Feature History Table

Feature Name	Release Information	Description
AI-driven telemetry (ADT)	Release 7.3.1	This feature leverages machine learning to detect and retrieve important network-state changes on the router. Relevant data is filtered and exported to the network management system for analysis or troubleshooting purposes. ADT significantly simplifies the configuration of streaming telemetry, and you are no longer required to manually choose sensor paths or tune the cadence at which counters have to be collected.

Cisco IOS XR offers a rich set of sensor paths that help you stream telemetry data about your network using Establish a Model-Driven Telemetry Session from a Router to a Collector. However, this rich offering of sensor paths leads to the following questions:

- Which sensor paths do I have to subscribe to?
- When do I know that the state of the system has changed, and which parameters do I look for troubleshooting purposes?
- How do I map between my router configuration and the sensor paths that I must monitor?
- Can I get the state changes in an automated, unsupervised, and data-driven way?
- Can I automatically filter a small set of sensor paths that properly portray an event?

AI-Driven Telemetry (ADT) is the answer to all these questions.

This article describes ADT, a component of the Cisco IOS XR operating system. ADT leverages machine learning (ML) and artificial intelligence (AI) to detect and describe important state changes on the router, and export only the relevant data using telemetry.

ADT eases streaming telemetry data and does not require the complex configurations of MDT. MDT supports streaming telemetry data at a configured Cadence-driven Telemetry, or when an Event-driven Telemetry occurs in the network. However, with MDT, not all telemetry data can be exported by the router at high frequencies. You must choose and subscribe to only a subset of the available information. Data availability does not mean that the data is easily consumable. You may miss some events generated within a cadence duration, either because you did not subscribe to the appropriate sensor paths or data models, or because the sampling rate was too coarse. On the other hand, you may be overwhelmed with redundant information, because a single event is represented through a large set of correlated counters.

ADT provides data-correlation and filtering on the router so that the router exports only the relevant data to network management systems (NMS). ADT correlates data by detecting important state changes on the router using a multivariate, unsupervised machine learning approach. ADT significantly simplifies the configuration of streaming telemetry. You are not required to choose the set of sensor paths (counters) to collect from the system, nor tune the cadence. ADT describes a change in the state of the router by choosing a small set of representative counters that best portray the change. The results of ADT change detection and description are provided as event-driven streaming telemetry using YANG model.

Consider a scenario where a bidirectional forwarding detection (BFD) session is enabled. Ideally, the system must filter a set of counters that are related to BFD. ADT uses automated feature selection process, wherein, in addition to streaming BFD-related counters, ADT might also highlight counters related to CPU. This is because BFD changes can trigger route re-calculation and eventually CPU occupancy. This shows that ADT learns the associated configuration in a system, and streams a representative set of sensor paths that get impacted with the state change. So, the uses cases for ADT falls under a broad spectrum where any event that causes fluctuation at the interface-level traffic is detected in an unsupervised manner.

With this streamed telemetry data, a data lake is created at the collector. Analyzing this data, you proactively monitor your network, identify patterns, troubleshoot your network in a predictive manner, and devise strategies to create a resilient network using automation.

- Key Components, on page 38
- Processing Telemetry Data on the Router to Analyze Traffic Changes, on page 39

Key Components

ADT comprises of the following key components:

- **Collector:** Collects all data from the router to provide a comprehensive view of the system. This function is synonymous to MDT.
- **Detector:** Prepares the collected data, remove redundancies, and detect changes to provide a macroscopic view of the system unlike Event Driven Telemetry (EDT). The detector learns how the sensor paths evolve to depend on each other and monitors changes at the system level. This learning about dependancies is unsupervised, multi-variate and adaptive to the data stream.
- **Selector:** Chooses a small, representative subset of the set of counters that best portrays events that are detected by the Detector using AI and unsupervised ML.
- **Exporter:** Streams the selected sensor paths using the ADT YANG data model to the MDT infrastructure. The ADT data model supports EDT where data is streamed only when an event in the system is detected. With sample interval `0`, there is no overhead due to high frequency push.

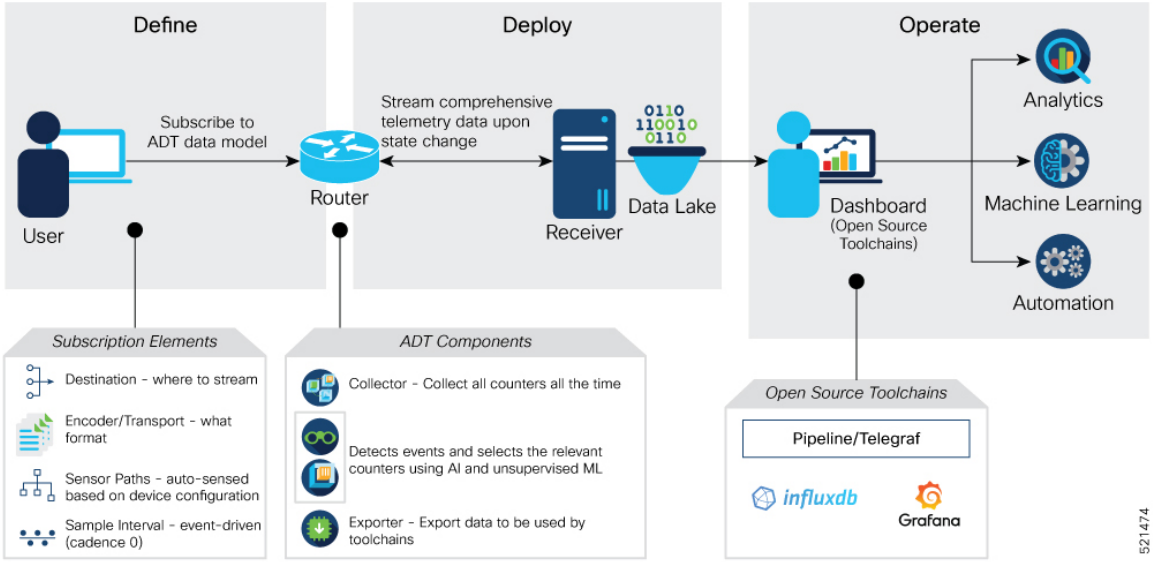
Processing Telemetry Data on the Router to Analyze Traffic Changes

The use case illustrates how, with the AI-driven telemetry (ADT), you can use telemetry data to monitor events in the network. Monitoring the events ensures efficient network management. This use case describes the tools that are 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:

Figure 12: AI-Driven Telemetry



- **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 analyze telemetry data using Open Source tools, and take necessary actions based on analysis.

Before you begin

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

Define a Subscription to Stream ADT Events from Router to Receiver

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

Step 1 Enable telemetry on the router.

Example:

```
Router(config)#telemetry model-driven
Router(config)#commit
```

Step 2 Enable ADT on the router.

Example:

```
Router(config)#adt enable
Router(config)#commit
```

Once ADT is enabled, it monitors a set of sensor paths that are derived from the configuration on the router. ADT learns these sensor paths in an unsupervised machine learning approach.

Note The `Cisco-IOS-XR-adt-config-cfg.yang` YANG data model can be used to configure ADT using NETCONF protocol.

```
container adt-config {
    description "Container Schema adt configuration";

    container enable-feature {
      presence "CLI submode compatibility.";
      description "Enable adt feature";

      container input {
        description "Sources for ADT collector";

        container mdt {
          description "MDT config for collector";

          container mdt-sensor-group-ids {
            description "MDT sensor groups";

            list mdt-sensor-group-id {
              key "groupname";
              description "MDT sensor group id";
              leaf groupname {
                type xr:Cisco-ios-xr-string;
                description "Mdt Sensor Group name";
              }
            }
          }
        }
      }
    }
  }
```

The following example shows the basic configuration to enable ADT using data model.

```
<config>
    <adt-config xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-adt-config-cfg">
        <enable-feature/>
    </adt-config>
</config>
```

To disable ADT, use the following operation:

```
<config>
    <adt-config xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-adt-config-cfg">
        <enable-feature xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
nc:operation="delete"/>
    </adt-config>
</config>
```

Step 3 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 fully qualified domain name (FQDN), port, transport, and encoding format in the destination-group.

Example:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group ADT-DESTINATION
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
```

Where -

- `ADT-DESTINATION` 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 hardcoding IP address, the router can choose any of the configured IPv4 or IPv6 address using domain name service. If an established connection 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. For the supported formats, see Encoder, on page 13.
- `tcp` is the protocol through which data is transported to the destination. For the supported protocols, see Transport, on page 13.

Step 4 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:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group ADT-EVENT
Router(config-model-driven-snsr-grp)#sensor-path Cisco-IOS-XR-adt-oper:adt/adt-output
```

Where -

- `ADT-EVENT` is the name of the sensor-group
- `Cisco-IOS-XR-adt-oper:adt/adt-output` is the sensor path from where data is streamed.

Note The following example showss how to add a custom sensor group using data model.

```
<config>
    <adt-config xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-adt-config-cfg">
        <enable-feature>
            <input>
                <mdt>
                    <mdt-sensor-group-ids>
                        <mdt-sensor-group-id>
                            <groupname> QOS_VOQ </groupname>
                        </mdt-sensor-group-id>
                    </mdt-sensor-group-ids>
                </mdt>
            </input>
        </enable-feature>
    </adt-config>
</config>
```

Step 5 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 Telemetry or Event-driven Telemetry. ADT is event-driven; subscriptions are defined with `0` (zero) cadence.

Example:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription ADT-SUBSCRIPTION
```

```
Router(config-model-driven-subs)#sensor-group-id ADT-EVENT sample-interval 0
Router(config-model-driven-subs)#destination-id ADT-DESTINATION
Router(config-model-driven-subs)#source-interface Interface1
```

Where -

- `ADT-SUBSCRIPTION` is the name of the subscription
- `ADT-EVENT` is the name of the sensor-group
- `ADT-DESTINATION` 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.
- `0` is the sample interval.

Running Configuration

```
adt enable
!
telemetry model-driven
 destination-group ADT-DESTINATION
  destination 172.0.0.0 port 57500
   encoding self-describing-gpb
   protocol tcp
  !
 !
 sensor-group ADT-EVENT
  sensor-path Cisco-IOS-XR-adt-oper:adt/adt-output
 !
 subscription ADT-SUBSCRIPTION
  sensor-group-id ADT-EVENT sample-interval 0
  destination-id ADT-DESTINATION
 !
!
```

Alternate Method: Configure ADT Using YANG Data Model

The above ADT configuration using CLI can also be configured using YANG data model `Cisco-IOS-XR-adt-config-cfg.yang`. You can obtain the data model from Github.

```
<config>
    <adt-config xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-adt-config-cfg">
        <enable-feature/>
    </adt-config>
</config>
```

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.

ADT operational models are classified into three categories:

- **ADT events:** This model defines how ADT generates output to describe an event using `Cisco-IOS-XR-adt-oper:adt/adt-output` sensor path.

- **ADT subscriptions:** This model exports the sensor paths that are monitored by ADT and the cadence at which each path is monitored using `Cisco-IOS-XR-adt-oper:adt/subscription-info` sensor path.

```
Router#show adt subscription details
Wed Feb  3 14:51:39.444 IST
ADT SUBSCRIPTION  Details
[Subscription ID, Cadence(in seconds), (E)xplicit/(I)mplicit] Sensor Path
*Subscription ID = 0: Not enough system resources to subscribe

Active Groups : 2

    Group: QOS_VOQ
    ----------------------------------------
        [300000028,  20, E]
Cisco-IOS-XR-qos-ma-oper:qos/interface-table/interface/output/service-policy-names/service-policy-instance/statistics

        [300000027,  20, E]
Cisco-IOS-XR-fretta-bcm-dpa-npu-stats-oper:dpa/stats/nodes/node/npu-numbers/npu-number/display/base-numbers/base-number


    Group: implicit
    ----------------------------------------
        [300000001,  60, I]
Cisco-IOS-XR-telemetry-model-driven-oper:telemetry-model-driven/subscriptions/subscription

        [300000002,  40, I]
Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
       [300000003,  40, I] Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary

        [300000004,  40, I]
Cisco-IOS-XR-procmem-oper:processes-memory/nodes/node[node-name="0/RP0/CPU0"]/process-ids/process-id[process-id="8893"]

        [300000005,  20, I]
Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/node-type-sets/node-type-set/interface-summary

        [300000006,  20, I]
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface[interface-name=".*(GigE|Ethernet).*"]/latest/generic-counters

        [300000007,  20, I] Cisco-IOS-XR-ipv6-io-oper:ipv6-io
        [300000008,  20, I] Cisco-IOS-XR-ip-iarm-v6-oper:ipv6arm/router-id

       --------------------------------- Truncated for Brevity -------------------------
```

- **ADT statistics:** This model reports health statistics of ADT system using `Cisco-IOS-XR-adt-oper:adt/statistics` sensor path.

In this example, the output is shown for ADT events. ADT event can have one or more events reported.

- Each event contains:
  - An event identifier
  - The time-stamp of the event
  - A short description of the event
  - List of sensors that changed their behavior during the event
- Each sensor path exported contains:
  - Tags which define the scope of the sensor path
  - List of value, timestamp pair containing samples of the sensor output before and after the event.

Step 1 View the generated events.

Example:

Following is a snippet of ADT event output, generated by traffic change.


```
  "node_id_str": "PE4",
  "subscription_id_str": "app_1887_75f00000001",
  "encoding_path": "Cisco-IOS-XR-adt-oper:adt/adt-output",
  "collection_id": "9569581",
  "collection_start_time": "1607525488535",
  "msg_timestamp": "1607525488556",
  "data_json": [
    {
      "timestamp": "1607525488552",
      "keys": [],
      "content": {
        "adt-event": [
          {
            "event-id": 123,
            "change-description": "Traffic",
            "timestamp": "1607431905419",
            "change": [
              {
                "sensor-path":
"Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters/bytes-received",

                "sensor-path-tags": "interface-name=GigabitEthernet0/3/0/19",
                "data": [
                  {
                    "value": {
                      "value-type": 8,
                      "val-counter64": "62808023132655"
                    },
                    "timestamp": "1607431545418"
                  },
                  ...
                  ...
                  {
                    "value": {
                      "value-type": 8,
                      "val-counter64": "62869633436614"
                    },
                    "timestamp": "1607432235421"
                  },
                  {
                    "value": {
                      "value-type": 8,
                      "val-counter64": "62872314602090"
                    },
                    "timestamp": "1607432265421"
                  }
                ]
              }
            ]
          }
        ]
      }
    }
  ],
  "collection_end_time": "1607525488556"
```

The router streams data to the receiver upon state change using the subscription-based telemetry session. A data lake is created at the receiver.

See the ADT events.

```
Router#show adt events
|-------------------------------------------------------------------------------|
| Number of Events :      5                                                     |
|-------------------------------------------------------------------------------|
|  Event id |                   Timestamp | Description                         |
|-------------------------------------------------------------------------------|
|       119 |  Tue 2020-12-01 13:41:56:141|                              Traffic|
|       120 |  Thu 2020-12-03 19:36:14:937|            Addressing & Reachability|
|       121 |  Thu 2020-12-03 20:00:48:015|            Addressing & Reachability|
|       122 |  Sun 2020-12-06 17:09:57:994|                              Traffic|
|       123 |  Tue 2020-12-08 18:21:45:419|                              Traffic|
|-------------------------------------------------------------------------------|
```

Step 2 See the details of events.

Example:

In this example, the details of event 123 are displayed.

```
Router#show adt events id 123 detail
Event Id             : 123
 Timestamp            : Tue 2020-12-08 18:21:45:419
 Description          : Traffic
 Number of Sensor paths : 1
    Sensor Path      :
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters/bytes-received

    Sensor Path Tags : interface-name=GigabitEthernet0/3/0/19
    Message          :
        Number of entries in list: 25
        Value : [ 62808023132655, 62810701566605, 62813380497605, 62816056633805,
                  62818733833405, 62821412539395, 62824092600551, 62826773125641,
                  62829461449196, 62832132687840, 62834805072011, 62837462182289,
                  62840165873427, 62842846468785, 62845523517431, 62848199893619,
                  62852053505122, 62853557475735, 62856237814551, 62858917998694,
                  62861594969436, 62864275943572, 62866948687442, 62869633436614,
                  62872314602090, ]
        First Timestamp : Tue 2020-12-08 18:15:45:418 [1607431545418]
        Last Timestamp  : Tue 2020-12-08 18:27:45:421 [1607432265421]
```

ADT event lists 25 sample values of data that are reported by sensor paths before and after the event. So, 25 data points help us describe a particular event and its associated sensor path.

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

You can start consuming and analysing 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 that is 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 (time series database (TSDB)) 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 that is 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 the 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 on an event change, and Telegraf requests information from the Pipeline at 1-second intervals.

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
```

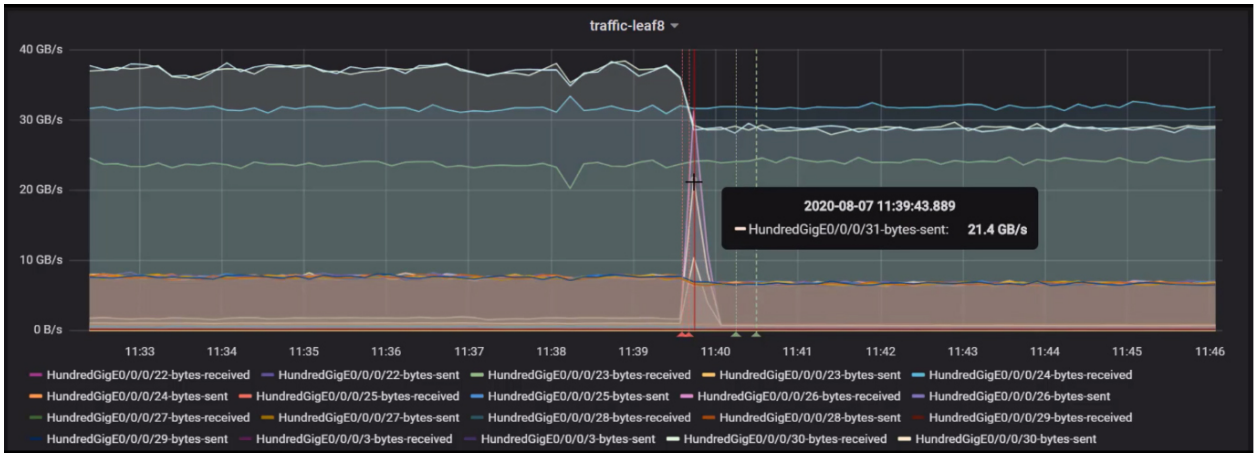
Step 2 In the Telegraph configuration file, add the following values to read the metrics about CPU usage.

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.

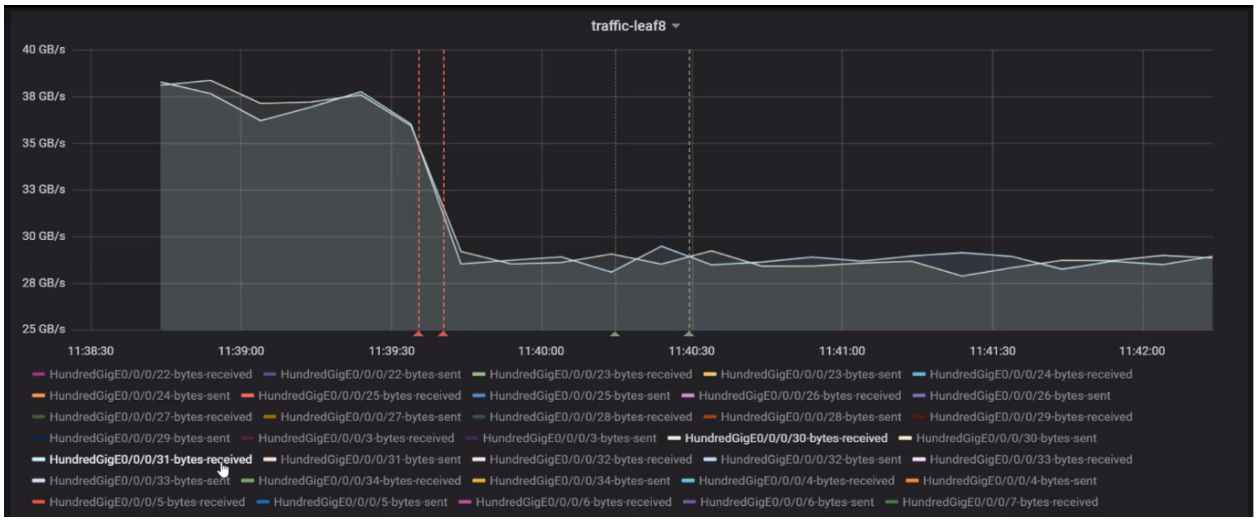
Figure 13: Visual Analysis of the Interfaces Impacted Due to Traffic Change Using Telemetry Data



521385

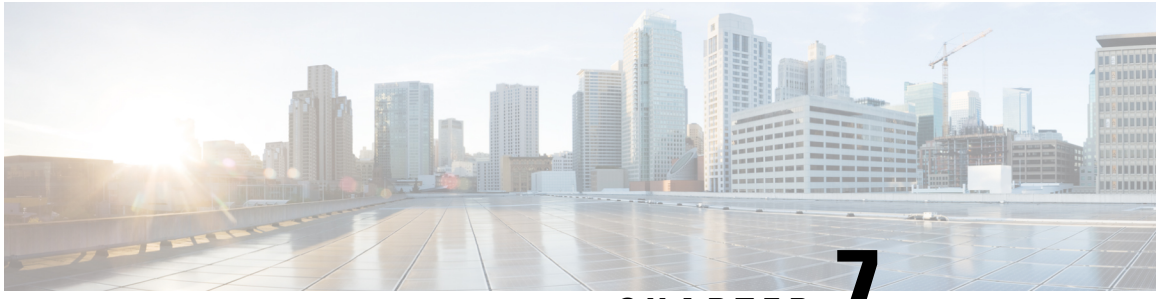
Further, narrow down to view the impact on individual interfaces. In this example, the bytes received for interfaces HundredGigE0/0/0/30 and HundredGigE0/0/0/31 are visualized and analysed.

Figure 14: Visual Analysis of System Monitoring on Two Interfaces Using Telemetry Data



521386

In conclusion, until the point where an event occurred, there is relatively no change. When an event is detected, there is significant drop in traffic on few interfaces and a peak on few other interfaces. ADT predicted these changes in the interfaces accurately using the associated sensor paths that it learned using AI and unsupervised ML from the router's configuration.



CHAPTER 7

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"}
```

Enhanced Syslog Notifications for Unresolved Line Card Forwarding Paths

Table 6: 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 7: Feature History Table

Feature Name	Release Information	Description
Target-Defined Mode for Cached Generic Counters Data	Release 7.5.1	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**

gNMI Dial-Out via Tunnel Service

Table 8: Feature History Table

Feature Name	Release Information	Description
gNMI Dial-Out via Tunnel Service	Release 7.6.1	This feature uses the tunnel service to allow the router (tunnel client) to dial out to a collector (tunnel server). After the session is established, the server-client switch directions where a server can act as a client to request gNMI services without altering the gNMI semantics. With this feature, the management software automatically learns when a new device is introduced in the network, thus saving you the manual, offline effort required to ensure the device connects to the management software. This feature introduces the keyword **tunnel** to the **grpc** command.

gNMI supports a dial-in session where a client connects to the router via gRPC server with the gNMI specification. This feature introduces support to use a tunnel service for gNMI dial-out connections based on the recommendation from OpenConfig forum.

With the gNMI dial-out through tunnel service, the router (tunnel client) dials out to a collector (tunnel server). Once the session is established, the tunnel server can act as a client and request gNMI services and gNMI Subscribe RPCs over the tunnel session. This feature allows a change in direction of session establishment and data collection without altering the gNMI semantics. Using gRPC tunnel dial-out session, the router initiates the connection to external collector so that the management software is automatically aware when a new device is introduced into the network.

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

Note Only the gNMI Subscribe RPC over the tunnel is supported.

Note The tunnel service supports only Transport Layer Security (TLS) session.

Perform the following steps to configure gNMI dial-out via tunnel service:

Step 1 Configure a third-party application (TPA) source address. This address sets a source hint for Linux applications, so that the traffic originating from the applications can be associated to any reachable IP (IPv4 or IPv6) address on the router.

Example:

```
Router(config)#tpa
Router(config)#vrf default
Router(config-vrf)#address-family ipv4
Router(config-vrf)#update-source dataports TenGigE0/6/0/0/1
```

A default route is automatically gained in the Linux shell.

Step 2 Configure the gNMI tunnel service on the router.

Example:

```
Router(config)#grpc tunnel destination ipv4
port 59510 source-interface TenGigE0/6/0/0/1 target Target-1 vrf default
```

Where—

- source-interface: Source ethernet interface
- target: Target name to register the tunnel service
- vrf: Virtual Routing and Forwarding (VRF) instance for the dial-out session. If VRF and source-interface are configured, VRF takes precedence over the source-interface.

Step 3 Verify that the gRPC tunnel configuration is successful on the router.

Example:

```
Router#show run grpc
Wed Nov 24 19:37:21.015 UTC
grpc
  port 57500
  no-tls
  tunnel
    destination 5.0.0.2 port 59510
      target Target-1
```

```
      source-interface GigabitEthernet0/0/0/1
    !
    destination 2002::1:2 port 59510
       source-interface GigabitEthernet0/0/0/0
    !
    destination 192.0.0.1 port 59500
    !
    destination 192.0.0.1 port 59600
    !
  !
!
```

Step 4 View the status of tunnel destination.

Example:

```
Router#show grpc tunnel sessions
Wed Nov 24 19:41:38.863 UTC
 5.0.0.2:59510
 Target:           Target-1
 Status:           Not connected
 Error:            Source Interface is down
 Source interface: GigabitEthernet0/0/0/1
 Source address:   5.0.0.1
 Source VRF:       default

 [2002::1:2]:59510
 Target:           Target-2
 Status:           Connected
 Source interface: GigabitEthernet0/0/0/0
 Source address:   2002::1:1
 Source VRF:       default
 Last Connected:   2021-11-24 19:41:23

 192.168.122.1:59500
 Target:           Target-2
 Status:           Connected
 Last Connected:   2021-11-24 19:40:15

 192.168.122.1:59600
 Target:           Target-2
 Status:           Not connected
 Error:            cert missing /misc/config/grpc/192.0.0.1:59600.pem
 Last Attempted:   2021-11-24 19:41:15
```

Step 5 Copy the public certificate for the collector to `/misc/config/grpc/<ip-addr>:<port>.pem` directory. The router uses this certificate to verify the tunnel server, and establish a dial-out session.

Step 6 Run the collector.

Virtual Address as the Source IP Address of gRPC Tunnel

Table 9: Feature History Table

Feature Name	Release Information	Description
Virtual Address as the Source IP Address of gRPC Tunnel	Release 7.10.1	You can now configure a virtual address as the source IP address of gRPC tunnel. Since the virtual address refers to whichever management interface is currently active, a route processor (RP) card fail-over is managed with ease and disruptions are minimized. Model-driven telemetry and Event-driven telemetry are supported. The feature introduces these changes: **CLI:** • The keywords **source ipv4 virtual address** and **source ipv6 virtual address** are introduced in the **gRPC tunnel** command.

A virtual IP address from the address family is used as the source IP address to establish a gRPC tunnel. A new tunnel client would be created and used to dial to the tunnel server with the source address as the configured virtual IP address. The Cisco IOS-XR gRPC server would listen on the established tunnel for incoming gNMI connections. You can configure the address-family of the virtual IP address to be used as the source address of a gRPC tunnel. The address-family can be IPv4, IPv6, or both. You can configure one IPv4 and one IPv6 virtual address per VRF in Cisco IOS-XR. Later, the virtual IP address of the specific VRF and the address-family shall be looked up and used as the source IP address to establish a gRPC tunnel.

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

Note If both virtual IP address and source-interface are configured, then virtual IP address takes precedence.

Configure Virtual Address as the Source IP Address of gRPC Tunnel

Perform the following steps to configure virtual IP as source address of gRPC tunnel:

Step 1 Configure virtual IP as source address of gRPC tunnel.

Example:

In this example, you set up a virtual IPv4 as source address.

```
Router#config

Router(config)#grpc
Router(config-grpc)#tunnel
Router(config-grpc-tunnel)#destination 192.168.0.1 port 59500
Router(config-grpc-tunnel-dest)#target xr
Router(config-grpc-tunnel-dest)#source ipv4 virtual address
Router(config-grpc-tunnel-dest)#source-interface MgmtEth 0/RP0/CPU0/0
Router(config-grpc-tunnel-dest)#commit
Tue Jul 25 05:29:35.933 UTC
Router(config-grpc-tunnel-dest)#end
```

In this example, you set up a virtual IPv6 as source address.

```
Router#config

Router(config)#grpc
Router(config-grpc)#tunnel
Router(config-grpc-tunnel)#destination 2001:DB8:A:B::1 port 59500
Router(config-grpc-tunnel-dest)#target xr
Router(config-grpc-tunnel-dest)#source ipv6 virtual address
Router(config-grpc-tunnel-dest)#source-interface MgmtEth 0/RP0/CPU0/0
Router(config-grpc-tunnel-dest)#commit
Tue Jul 25 05:30:46.104 UTC
Router(config-grpc-tunnel-dest)#end
```

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

```
Router#show running-config grpc

grpc
tunnel
  destination 192.168.0.1 port 59500
   target xr
   source ipv4 virtual address
   source-interface MgmtEth0/RP0/CPU0/0
  !
  destination 2001:DB8:A:B::1 port 59500
   target xr
   source ipv6 virtual address
   source-interface MgmtEth0/RP0/CPU0/0
  !
!
!
```

Step 3 Verify the configured virtual IP address.

The virtual address takes effect once the VRF contains a management interface. Ensure the virtual address is on the same network as the management IP addresses.

Example:

The following example shows how to display the configured virtual IPv4 address:

```
Router#show ipv4 virtual address status vrf all

VRF Name: default
Virtual IP: 192.168.0.2/24
Active Interface Name: MgmtEth0/RP0/CPU0/0
Active Interface MAC Address: 02bf.4615.55aa
```

```
VRF Node Create Timestamp   : .1022
ARP Add Timestamp           : .1042
RIB Add Timestamp           : .1042
SNMAC Add Timestamp         : N/A
```

The following example shows how to display the configured virtual IPv6 address:

```
Router#show ipv6 virtual address status vrf all

VRF Name: default
Virtual IP: 2001:DB8:A:B::11/64
Active Interface Name: MgmtEth0/RP0/CPU0/0
Active Interface MAC Address: 02bf.4615.55aa

VRF Node Create Timestamp   : .8426
ND Add Timestamp            : .8626
RIB Add Timestamp           : .8526
SNMAC Add Timestamp         : .8626
```

Step 4 Copy the public certificate for the collector to `/misc/config/grpc/<ipv4 address>:<port>.pem` or `/misc/config/grpc/[<ipv6 address>]:<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

192.168.0.1:59500
Target:           xr
Status:           Connected
Source address:   192.168.0.2
Source VRF:       default
Virtual IP status: Active
Last connected:   2023-02-07 03:05:14

[2001:DB8:A:B::1]:59500
Target:           xr
Status:           Connected
Source address:   2001:DB8:A:B::11
Source VRF:       default
Virtual IP status: Active
Last connected:   2023-02-07 03:11:15
```

Stream Telemetry Data about PBR Decapsulation Statistics

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 NCS 5500 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.

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 ACL

Table 10: Feature History Table

Feature Name	Release Information	Description
Stream Telemetry Data for ACL Byte Counters	Release 7.9.1	You can stream model-driven telemetry (MDT) data to monitor the ACL statistics such as dropped, matched and denied IPv4 and IPv6 packets using `Cisco-IOS-XR-ipv4-acl-oper.yang` and `Cisco-IOS-XR-ipv6-acl-oper.yang` data models. This release lets you stream telemetry data to monitor the statistics using byte counters. Previously, the only option to monitor ACL statistics was to use packet counters. ACL with policer statistics is supported only on Cisco Network Convergence System 5700 Series Routers.

Feature Name	Release Information	Description
Stream Telemetry Data for ACL	Release 7.8.1	The Access control List (ACL) is an ordered list of rules used to filter the traffic to increase network performance, and to specify the system resource access permissions either grant or deny to users or systems for security. We have introduced the streaming of ACL statistics to monitor the traffic flow using YANG data and telemetry. It allows you to monitor dropped, matched, and denied packets of IPv4 and IPv6. In earlier releases, you could monitor ACL statistics through CLI. This feature introduces the `Cisco-IOS-XR-ipv4-acl-oper.yang` and `Cisco-IOS-XR-ipv6-acl-oper.yang` models to capture IPv4 and IPv6 ACL statistics on Cisco Network Convergence System 5700 Series Routers.

Prior to Cisco IOS XR Software Release 7.8.1, ACL statistics were viewed using **show run ipv4 access-list** and **show run ipv6 access-list** commands. From Cisco IOS XR Software Release 7.8.1 you can stream telemetry data for ACL statistics using `Cisco-IOS-XR-ipv4-acl-oper.yang` and `Cisco-IOS-XR-ipv6-acl-oper.yang` data models.

For more information on ACL, see *Interface and Hardware Component Configuration Guide for Cisco NCS 5500 Series Routers*.

You can stream ACL telemetry data from the following XPaths:

```
Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/
access-list-manager/accesses/access/access-list-sequences/access-list-sequence
```

```
Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/
access-list-manager/accesses/access/access-list-sequences/access-list-sequence
```

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

SUMMARY STEPS

1. Check the configuration of ACL packets for IPv4 and IPv6.
2. View ACL statistics for IPv4 and IPv6 (Ingress or Egress direction-wise).
3. View Model Driven Telemetry (MDT) of ACL statistics.

DETAILED STEPS

Step 1 Check the configuration of ACL packets for IPv4 and IPv6.

Example:

```
Router# show run ipv4 access-list
ipv4 access-list test
 10 permit tcp any any
 20 deny udp any any
!
ipv4 access-list tempv4
 10 deny udp any port-group p1 any
```

```
 20 deny tcp any any
!
```

Example:

```
Router# show run ipv6 access-list
Thu Jun 16 18:03:29.864 UTC
ipv6 access-list v6
 10 permit tcp any any
 20 deny udp any any
!
ipv6 access-list tempv6
 10 deny udp any port-group p1 any
 20 deny tcp any any
!
```

Step 2 View ACL statistics for IPv4 and IPv6 (Ingress or Egress direction-wise).

Example:

```
Router# show access-lists ipv4 tempv4 hardware ingress location 0/1/CPU0
ipv4 access-list tempv4
 10 deny udp any port-group p1 any (83319 matches)
 20 deny tcp any any (83319 matches)
```

Example:

```
Router# show access-lists ipv6 tempv6 hardware ingress location 0/1/CPU0
ipv6 access-list tempv6
 10 deny udp any port-group p1 any (55792 matches)
 20 deny tcp any any (55792 matches)
!
```

Step 3 View Model Driven Telemetry (MDT) of ACL statistics.

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 Model-Driven Telemetry collectors, see Establish a Model-Driven Telemetry Session from a Router to a Collector, on page 19.

Example:

MDT of ACL IPv4 statistics

```
Router# run mdt_exec -s Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/
access-list-manager/accesses/access/access-list-sequences/access-list-sequence -c 30000
Enter any key to exit...
Request datatree:
    filter
        ipv4-acl-and-prefix-list (ka)
            access-list-manager
                accesses
                    access
                        access-list-sequences
                            access-list-sequence
Sub_id 200000001, flag 0, len 0
Sub_id 200000001, flag 4, len 6739
--------
{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001",
"encoding_path":"Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/
accesses/access/access-list-sequences/access-list-sequence","collection_id":"1",
"collection_start_time":"1655427578624","msg_timestamp":"1655427578632",
"data_json":[{"timestamp":"1655427578629","keys":[{"access-list-name":"tel_test"},
{"sequence-number":10}],"content":{"item-type":"normal","sequence":10,"grant":"permit",
"protocol-operator":0,"protocol":512,"protocol2":0,"source-address":"0.0.0.0","source-address-mask":"255.255.255.255"
--------
```

```
"fragment-offset1":0,"fragment-offset2":0,"set-ttl":65535,"fragment-flags":0,"police":{"police-value":0,
"police-unit":"pps","police-peak-value":0,"police-peak-unit":"pps"},"priority":"acl-priority-unspec",
"is-icmpon":false}}],"collection_end_time":"1655427578633"}
--------
```

Example:

MDT of ACL IPv6 statistics:

```
Router# run mdt_exec -s
Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/
access/access-list-sequences/access-list-sequence -c 30000
Enter any key to exit...
Request datatree:
    filter
        ipv6-acl-and-prefix-list (ka)
            access-list-manager
                accesses
                    access
                        access-list-sequences
                            access-list-sequence
Sub_id 200000001, flag 0, len 0
Sub_id 200000001, flag 4, len 4005
--------
{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001","encoding_path":
"Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/
access/access-list-sequences/access-list-sequence","collection_id":"1",
"collection_start_time":"1655432482881","msg_timestamp":"1655432482886",
"data_json":[{"timestamp":"1655432482884","keys":[{"access-list-name":"test"},
{"sequence-number":10}],"content":{"is-ace-type":"normal","is-ace-sequence-number":10,
"is-packet-allow-or-deny":"permit","is-protocol-operator":"none",
"is-ipv6-protocol-type":6,"is-ipv6-protocol2-type":0,"is-source-address-in-numbers":
....
"police-peak-unit":"pps"},"priority":"acl-priority-unspec","fragment-flags":0,
"is-icmp-message-on":0}}],"collection_end_time":"1655432482886"}
--------
Sub_id 200000001, flag 8, len 0
Sub_id 200000001, flag 4, len 4005
```

You can apply filter on ACL name as followed:

```
Router# run mdt_exec -s
Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/accesses/
access[access-list-name="test"]/access-list-sequences/access-list-sequence -c 30000
```

Stream Telemetry Data for ACL Byte Counters

You can subscribe to the following options based on the requirement:

- Stream entire ACL data:

  Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/accesses/access/access-list-sequences/access-list-sequence

  Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/access/access-list-sequences/access-list-sequence

- Stream specific ACL data:

  ```
  Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/accesses/access
  [access-list-name="ipv4_permit_tcp_any_any"]/access-list-sequences/access-list-sequence
  ```

```
Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/access
[access-list-name="ipv6-check-traditional"]/access-list-sequences/access-list-sequence
```

- Stream specific sequence of ACL data:

```
Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/accesses/access
[access-list-name="ipv4_permit_tcp_any_any"]/access-list-sequences/access-list-sequence[sequence-number=20]

Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/access
[access-list-name="ipv6-check-traditional"]/access-list-sequences/access-list-sequence[sequence-number=20]
```

View the telemetry subscription with the ACL sensor paths.

```
Router#show run telemetry model-driven
Mon Mar 27 09:49:48.158 UTC
telemetry model-driven
destination-group tcam
  address-family ipv4 5.10.14.20 port 8000
  encoding json
  protocol tcp

sensor-group tcam
  sensor-path
Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/accesses/access/access-list-sequences/access-list-sequence

  sensor-path
Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/access/access-list-sequences/access-list-sequence

subscription tcam
  sensor-group-id fs sample-interval 300000
  destination-id tcam
 !
!
```

View the byte counters configured for ACL statistics.

The following example shows the statistics for IPv4 address family:

```
Router#show access-lists ipv4 v4 hardware ingress location 0/7/CPU0
Thu Mar 30 06:16:17.819 UTC
ipv4 access-list v4
 10 permit ipv4 any 2.2.0.0 0.0.255.255 dscp af11 (11035388 matches) (1368388112 bytes)
```

The following example shows the statistics for IPv6 address family:

```
Router#show access-lists ipv6 v6 hardware ingress location 0/7/CPU0
Thu Mar 30 06:16:54.723 UTC
ipv6 access-list v6
 10 permit ipv6 any 2222::/64 dscp af11 (11035388 matches) (1368388112 bytes)
```

After you have verified that the statistics are displayed correctly, stream the telemetry data. You can view the streamed data at the collector.

MDT of ACL IPv4 statistics:

```
Router#run mdt_exec -s
"Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/accesses/access
[access-list-name=="v4"]/access-list-sequences/access-list-sequence[sequence-number=10]"
-c 300000
Thu Mar 30 06:19:06.698 UTC
Enter any key to exit...
 Sub_id 200000001, flag 0, len 0
 Sub_id 200000001, flag 4, len 5452
--------
```

{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001","encoding_path":"Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/accesses/access/access-list-sequences/access-list-sequence","collection_id":"1","collection_start_time":"1663309156852","msg_timestamp":"1663309156862","data_json":[{"timestamp":"1663309156860","keys":[{"access-list-name":"v4"},{"sequence-number":10}],"content":{"item-type":"normal","sequence":10,"grant":"permit","protocol-operator":0,"protocol":512,"protocol2":0,"source-address":"0.0.0.0","source-address-mask":"255.255.255.255","destination-address":"2.2.0.0","destination-address-mask":"0.0.255.255","source-operator":"none","source-port1":0,"source-port2":0,"destination-operator":"none","destination-port1":0,"destination-port2":0,"log-option":"log-none","capture":false,"dscp-present":true,"dscp":10,"dscp2":0,"dscp-operator":0,"dscp-bitmask":255,"precedence-present":false,"precedence":0,"tcp-flags-operator":"match-none","tcp-flags":0,"tcp-flags-mask":0,"fragments":0,"packet-length-operator":"none","packet-length1":0,"packet-length2":0,"ttl-operator":"none","ttl1":0,"ttl2":0,"no-stats":false,"hits":"3","hardware-hits":"11035388","police-hits":"0","police-hits-dropped":"0","byte-hits":"1368388112","byte-police-hits":"0","byte-police-hits-dropped":"0","is-icmp-off":false,"qos-group":65535,"next-hop-type":"nexthop-none","next-hop-info":[{"next-hop":"0.0.0.0","status":"not-present","at-status":"unknown","is-acl-next-hop-exist":false},{"next-hop":"0.0.0.0","status":"not-present","at-status":"unknown","is-acl-next-hop-exist":false},{"next-hop":"0.0.0.0","status":"not-present","at-status":"unknown","is-acl-next-hop-exist":false}],"dynamic":false,"acl-name":"v4","sequence-str":"10","fragment-offset-operator":"none","fragment-offset1":0,"fragment-offset2":0,"set-ttl":65535,"fragment-flags":0,"police":{"police-value":0,"police-unit":"pps","police-peak-value":0,"police-peak-unit":"pps"},"priority":"acl-priority-unspec",
"is-icmpon":false}}

MDT of ACL IPv6 statistics:

```
Router#run mdt_exec -s
"Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/
access[access-list-name="v6"]/access-list-sequences/access-list-sequence[sequence-number=10]"
 -c 300000
Thu Mar 30 06:19:41.464 UTC
Enter any key to exit...
 Sub_id 200000001, flag 0, len 0
 Sub_id 200000001, flag 4, len 6247
--------
```

{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001","encoding_path":"Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/access/access-list-sequences/access-list-sequence","collection_id":"3","collection_start_time":"1663309191611","msg_timestamp":"1663309191620","data_json":[{"timestamp":"1663309191618","keys":[{"access-list-name":"v6"},{"sequence-number":10}],"content":{"is-ace-type":"normal","is-ace-sequence-number":10,"is-packet-allow-or-deny":"permit","is-protocol-operator":"none","is-ipv6-protocol-type":513,"is-ipv6-protocol2-type":0,"is-source-address-in-numbers":"::","is-source-address-prefix-length":0,"source-mask":"::","is-destination-address-in-numbers":"2222::","is-destination-address-prefix-length":64,"destination-mask":"ffff:ffff:ffff:ffff::","is-source-operator":"none","is-source-port1":0,"is-source-port2":0,"is-destination-operator":"none","is-destination-port1":0,"is-destination-port2":0,"is-log-option":"log-none","is-tcp-bits-operator":"match-none","is-tcp-bits":0,"is-tcp-bits-mask":0,"is-dscp-present":1,"dscp-operator":0,"is-dscp-valu":10,"is-dscp-valu2":0,"dscp-bitmask":255,"is-precedence-present":0,"is-precedence-value":0,"is-header-matches":0,"is-packet-length-operator":"none","is-packet-length-start":0,"is-packet-length-end":0,"is-time-to-live-operator":"none","is-time-to-live-start":0,"is-time-to-live-end":0,"no-stats":0,"hits":"2","hardware-hits":"11035388","police-hits":"0","police-hits-dropped":"0","byte-hits":"1368388112","byte-police-hits":"0","byte-police-hits-dropped":"0","capture":0,"undetermined-transport":0,"is-icmp-message-off":0,"qos-group":65535,"next-hop-type":"nexthop-none","next-hop-info":[{"next-hop":"::","vrf-name":"","track-name":"","status":"not-present","at-status":"unknown","acl-nh-exist":0},{"next-hop":"::","vrf-name":"","track-name":"","status":"not-present","at-status":"unknown","acl-nh-exist":0},{"next-hop":"::","vrf-name":"","track-name":"","status":"not-present","at-status":"unknown","acl-nh-exist":0}],"is-flow-id":4294967295,"acl-name":"v6","sequence-str":"10","set-ttl":65535,"police":{"police-value":0,"police-unit":"pps","police-peak-value":0,"police-peak-unit":"pps"},"priority":"acl-priority-unspec","fragment-flags":0,"is-icmp-message-on":0}}

Stream Telemetry Data for BGP FlowSpec

Table 11: Feature History Table

Feature Name	Release Information	Description
Stream Telemetry Data for BGP FlowSpec Statistics	Release 7.8.1	Use Border Gateway Protocol (BGP) FlowSpec to mitigate the effects of distributed denial-of-service (DDoS) attack over the network. We have introduced streaming of BGP FlowSpec statistics using YANG data and telemetry. It allows you to monitor traffic flow match, drop in the traffic, or policing at definite rate for IPv4 and IPv6 parameters such as IP address, port, DSCP, and so on. In earlier releases, you could monitor BGP FlowSpec statistics through CLI. This feature introduces the `Cisco-IOS-XR-flowspec-oper.yang` data models to capture BGP FlowSpec statistics such as matched, dropped, and transmitted packet count on Cisco Network Convergence System 5700 Series Routers.

Prior to Cisco IOS XR Software Release 7.8.1, BGP FlowSpec statistics were viewed using **show flowspec vrf all afi-all detail statistics** command. From Cisco IOS XR Software Release 7.8.1 you can stream telemetry data for BGP FlowSpec statistics using a `Cisco-IOS-XR-flowspec-oper.yang` data model.

For more information on BGP FlowSpec, see *BGP Configuration Guide for Cisco NCS 5500 Series Routers*.

You can stream BGP FlowSpec telemetry data from the XPath:
`Cisco-IOS-XR-flowspec-oper:flow-spec/vrfs/vrf/afs/af/flows/flow`

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

SUMMARY STEPS

1. Check the configuration of the BGP FlowSpec.
2. View BGP FlowSpec statistics for IPv4 and IPv6.
3. View Model Driven Telemetry (MDT) of BGP FlowSpec statistics.

DETAILED STEPS

Step 1 Check the configuration of the BGP FlowSpec.

Example:

```
Router# show running-config
Client config:
router bgp 100
nsr
```

```
bgp router-id 2.2.2.1
address-family ipv4 unicast
!
address-family vpnv4 unicast
!
address-family ipv6 unicast
!
address-family vpnv6 unicast
!
address-family ipv4 flowspec
!
address-family ipv6 flowspec
!
address-family vpnv4 flowspec
!
address-family vpnv6 flowspec
!
neighbor 1.1.1.1
  remote-as 100
  update-source Loopback1
  address-family ipv4 unicast
  !
  address-family vpnv4 unicast
  !
  address-family ipv4 flowspec
  !
  address-family vpnv4 flowspec
  !
!
neighbor 1.1.1.2
 remote-as 100
  update-source Loopback2
  address-family ipv6 unicast
  !
  address-family vpnv6 unicast
  !
  address-family ipv6 flowspec
  !
  address-family vpnv6 flowspec
  !
!
!
flowspec
local-install interface-all
address-family ipv4
  local-install interface-all
  service-policy type pbr redirect
!
!
end

class-map type traffic match-all c1
match protocol sctp
 end-class-map
!
!
class-map type traffic match-all c2
match protocol udp
 end-class-map
!
class-map type traffic match-all c3
match dscp 3
 end-class-map
!
```

```
class-map type traffic match-all c1_6
match dscp af11
 end-class-map
!
class-map type traffic match-all c2_6
match dscp 20
 end-class-map
!
policy-map type pbr p1
class type traffic c1
  drop
!
 class type traffic c2
  drop
!
 class type traffic c3
  drop
!
 class type traffic class-default
 !
 end-policy-map
!
policy-map type pbr p1_6
class type traffic c1_6
  set dscp af21
!
 class type traffic c2_6
  set dscp af22
!
 class type traffic class-default
 !
 end-policy-map
!
router bgp 100
nsr
bgp router-id 1.1.1.1
address-family ipv4 unicast
!
address-family vpnv4 unicast
!
address-family ipv6 unicast
!
address-family vpnv6 unicast
!
address-family ipv4 flowspec
!
address-family ipv6 flowspec
!
address-family vpnv4 flowspec
!
address-family vpnv6 flowspec
!
neighbor 2.2.2.1
  remote-as 100
  update-source Loopback1
  address-family ipv4 unicast
  !
  address-family vpnv4 unicast
  !
  address-family ipv4 flowspec
  !
  address-family vpnv4 flowspec
  !
!
```

```
neighbor 2.2.2.2
  remote-as 100
  update-source Loopback2
  address-family ipv6 unicast
  !
  address-family vpnv6 unicast
  !
  address-family ipv6 flowspec
  !
  address-family vpnv6 flowspec
  !
!
!
flowspec
address-family ipv4
  service-policy type pbr p1
!
address-family ipv6
  service-policy type pbr p1_6
!
!
```

Step 2 View BGP FlowSpec statistics for IPv4 and IPv6.

Example:

```
Router# show flowspec vrf all afi-all detail statistics
AFI: IPv4
  Flow           :Proto:=17
Flowspec Rule:
    Matches:
      Protocol            :                   17
    Actions      :Traffic-rate: 0 bps  (bgp.1)
    Statistics                        (packets/bytes)
      Matched             :                   0/0
      Transmitted         :                   0/0
      Dropped             :                   0/0
  Flow           :Proto:=132
Flowspec Rule:
    Matches:
      Protocol            :                   132
    Actions      :Traffic-rate: 0 bps  (bgp.1)
    Statistics                        (packets/bytes)
      Matched             :                   0/0
      Transmitted         :                   0/0
      Dropped             :                   0/0
  Flow           :DSCP:=3
Flowspec Rule:
    Matches:
      DSCP                :                   3
    Actions      :Traffic-rate: 0 bps  (bgp.1)
   Statistics                        (packets/bytes)
      Matched             :                   0/0
      Transmitted         :                   0/0
      Dropped             :                   0/0


AFI: IPv6
  Flow           :DSCP:=10
Flowspec Rule:
    Matches:
      DSCP                :                   10
    Actions      :DSCP: af21  (bgp.1)
    Statistics                        (packets/bytes)
      Matched             :                   0/0
```

```
      Transmitted         :                   0/0
      Dropped             :                   0/0
  Flow           :DSCP:=20
Flowspec Rule:
    Matches:
      DSCP                :                   20
    Actions      :DSCP: af22  (bgp.1)
    Statistics                        (packets/bytes)
      Matched             :                   0/0
      Transmitted         :                   0/0
      Dropped
```

Step 3 View Model Driven Telemetry (MDT) of BGP FlowSpec statistics.

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 Model-Driven Telemetry collectors, see Establish a Model-Driven Telemetry Session from a Router to a Collector, on page 19.

Example:

MDT of BGP FlowSpec statistics

```
Router# run mdt_exec -s Cisco-IOS-XR-flowspec-oper:flow-spec/vrfs/vrf/afs/af/flows/flow
Enter any key to exit...
Request datatree:
    filter
        flow-spec (ka)
            vrfs
                vrf
                    afs
                        af
                            flows
                                flow
Sub_id 200000001, flag 0, len 0
Sub_id 200000001, flag 4, len 3952
--------
{"node_id_str":"PE","subscription_id_str":"app_TEST_200000001",
"encoding_path":"Cisco-IOS-XR-flowspec-oper:flow-spec/
vrfs/vrf/afs/af/flows/flow","collection_id":"2",
"collection_start_time":"1661410086614","msg_timestamp":"1661410086633",
...
"dscp":[{"min":20,"max":20}],"fragment-type":0,"tcp-flag":{"value":0,
"match-any":false}}}}],"collection_end_time":"1661410086635"}
-----------------------------------------------------
Sub_id 200000001, flag 8, len 0
```

View Internal TCAM Resource Utilization for Ingress Hybrid ACL

Table 12: Feature History Table

Feature Name	Release Information	Description
View Internal TCAM Resource Utilization for Ingress Hybrid ACL	Release 7.8.1	You can now fetch the usage data through CLI and Streaming Telemetry. Ternary Content-Addressable Memory (TCAM) is an important and limited resource. This feature, allows you to be mindful of the usage and availability of the resource, before configuring ingress hybrid ACL. This functionality modifies the following: • **CLI:** The option **status** in the **show controllers npu internaltcam status location** command, displays the possible free and used entries. • **YANG Data Model:** This feature uses the `Cisco-IOS-XR-fia-internal-tcam-oper.yang` to fetch the internal TCAM resource.

Internal TCAM is a valuable and constrained resource in hardware, which multiple features must share. A switch uses TCAM to store rules of various applications such as Quality of Service (QoS), Access Control Lists (ACLs), IP route tables, and VLANs. TCAM stores these rules in memory differently than normal memory RAM storage, where the IOS uses a memory address to search for specific data. TCAM, instead, uses the data first, and then it looks for the respective memory location. This way, the switch searches for these rules faster, improving overall performance.

If exhaustion of internal TCAM resource occurs, then a warning message is displayed. Further, while trying to configuring a new ACL, the hardware displays an error message. Thus, with the unavailability of resource, more ACL cannot be programmed. This impacts the network security and also causes poor performance.

Since, TCAM is a limited resource, it requires continuous monitoring. During the programming of ACL, using the **show** command to check the current TCAM utilization is helpful. As a result of this ability to perform a lookup simultaneously, you can avert any performance degradation.

In Cisco IOS XR Software Release 7.8.1, this feature lets you know the internal TCAM utilization and the available free space before configuring a hybrid ACL. Also, you can avoid over utilization of the resource.

View TCAM Usage

Display data using CLI

A new option **status** is added to the existing **show controllers npu** command, which displays the internal TCAM resources used by different features and number of possible entries the feature can further use.

The following is an example displaying the internal TCAM resource utilization and the possible free entries for hybrid ACL feature:

```
Router#show controllers npu internaltcam status location 0/0/CPU0
Thu Mar 24 12:17:49.224 UTC
Ingress TCAM Resource Usage Information
==============================
NPU Feature   Used    Free
================================
0   V4_ACL    2       8150
0   V6_ACL    40      8150
1   V4_ACL    0       8152
1   V6_ACL    40      8152
```

Display data using Streaming Telemetry

You can stream the data to the client using MDT (Model Driven Telemetry) along with gRPC protocol. The client needs to be subscribed to one of the subscription offered by the router to get the data from the router. This subscription is achieved by either Dial-In or Dial-Out method.

The stream of data is sent in intervals to the client through the Management interface.

You can stream the **free** TCAM resource telemetry data from the following XPath:

```
Cisco-IOS-XR-fia-internal-tcam-oper:controller/dpa/nodes/node/internal-tcam-resources/npu-tcam/tcam-usage/tcam-entries-free
```

You can stream the **used** TCAM resource telemetry data from the following XPath:

```
Cisco-IOS-XR-fia-internal-tcam-oper:controller/dpa/nodes/node/internal-tcam-resources/npu-tcam/tcam-usage/tcam-entries-used
```

Configuration Example

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.

The following example shows the configuration to create the subscription for internal TCAM resources and sensor path:

```
Router#show running-config telemetry model-driven
Tue Mar 22 15:06:17.516 UTC
telemetry model-driven
 sensor-group SGroup3
  sensor-path
Cisco-IOS-XR-fia-internal-tcam-oper:controller/dpa/nodes/node/internal-tcam-resources
 !
 subscription Sub3
  sensor-group-id SGroup3 sample-interval 3000
  source-interface MgmtEth0/RP0/CPU0/0
 !
!
```