



Telemetry Configuration Guide for Cisco NCS 540 Series Routers, IOS XR Release 7.3.x

First Published: 2021-02-26

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

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <https://www.cisco.com/c/en/us/about/legal/trademarks.html>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2021 Cisco Systems, Inc. All rights reserved.



CONTENTS

CHAPTER 1

Scale-Up Your Network Monitoring Strategy Using Telemetry 1

Benefits of Shifting Network Monitoring from Pull Models to Telemetry Push Model 3

Review Mechanisms to Stream Telemetry Data from a Router to a Destination 3

 Cadence-driven Telemetry 4

 Event-driven Telemetry 4

Learn About the Elements that Enable Streaming Telemetry Data 5

 Sensor Path 5

 Subscription 7

 Encoder 8

 Transport 8

 gRPC Network Management Interface 9

 gRPC Network Operations Interface 10

 TLS Authentication For Dial-in 10

Explore the Methods to Establish a Telemetry Session 11

 Dial-Out Mode 11

 Dial-In Mode 11

 Identify the Telemetry Session Suitable for Your Network 11

CHAPTER 2

Establish a Model-Driven Telemetry Session from a Router to a Collector 13

Monitor CPU Utilization Using Telemetry Data to Plan Network Infrastructure 14

 Define a Subscription to Stream Data from Router to Receiver 15

 Verify Deployment of the Subscription 18

 Operate on Telemetry Data for In-depth Analysis of the Network 19

CHAPTER 3

Establish a Model-Driven Telemetry Session from a Collector to a Router 23

Monitor Network Parameters Using Telemetry Data for Proactive Analysis 24

Define a Subscription to Stream Data from Router to Receiver 26

Verify Deployment of the Subscription 27

Operate on Telemetry Data for In-depth Analysis of the Network 28

CHAPTER 4

Hardware Timestamp 31

Target-Defined Mode for Cached Generic Counters Data 32



CHAPTER 1

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?

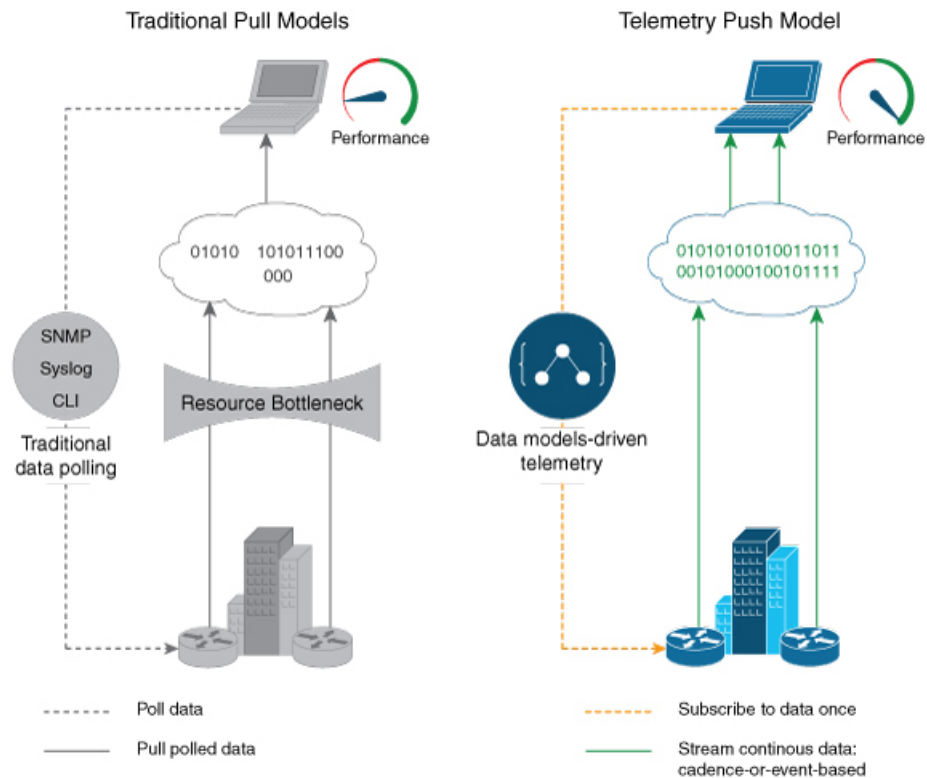
These traditional methods use a *pull* model to request information at regular intervals. The data that you collect may help you to efficiently monitor your network of a manageable size. However, as your network grows in complexity and scale, the data that you poll may be insufficient for efficient and 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, on page 3](#)
- [Review Mechanisms to Stream Telemetry Data from a Router to a Destination, on page 3](#)
- [Learn About the Elements that Enable Streaming Telemetry Data , on page 5](#)
- [Explore the Methods to Establish a Telemetry Session, on page 11](#)

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

Real-time telemetry data is useful in:

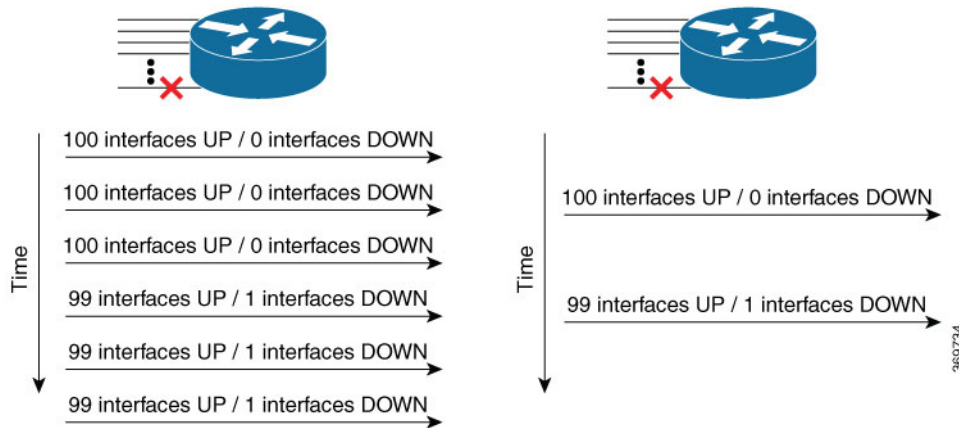
- **Managing network remotely:** The primary benefit of telemetry is the ability it offers you as an end user to monitor the state of a network element remotely. After the network is deployed, you cannot be physically present at the network site to find out what works, and what is cumbersome. With telemetry,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

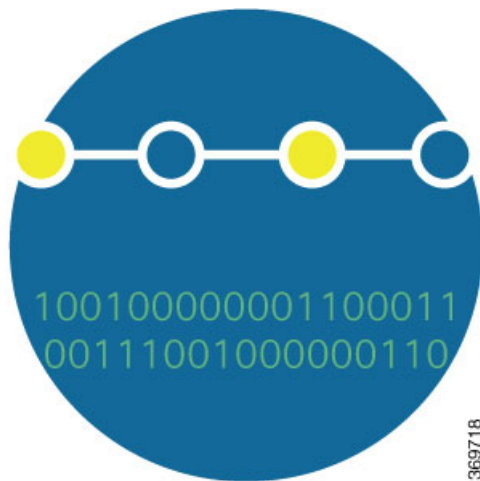


Cadence-driven Telemetry

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

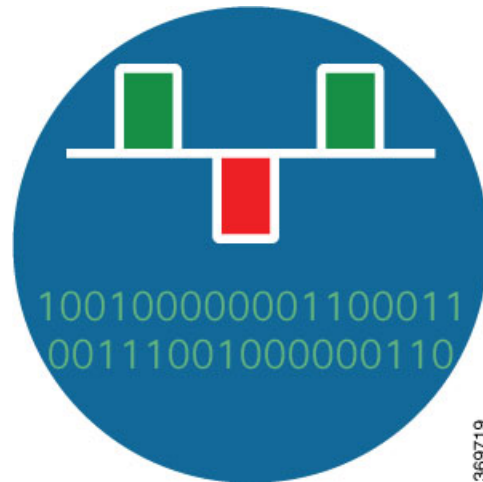


Event-driven Telemetry

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

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

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

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

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

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

The following table shows few examples of sensor paths:

Table 1: Sensor Paths

Feature	Sensor Path
CPU	Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization

Feature	Sensor Path
Memory	<code>Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary</code>
Interface	<code>Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters</code> <code>Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/data-rate</code> <code>openconfig-interfaces:interfaces/interface</code>
Node summary	<code>Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary</code>
Forwarding information base (FIB)	<code>Cisco-IOS-XR-fib-common-oper:fib-statistics/nodes/node/drops</code> <code>Cisco-IOS-XR-fib-common-oper:fib/nodes/node/protocols/protocol/vrfs/vrf/summary</code>
MPLS Traffic engineering (MPLS-TE)	<code>Cisco-IOS-XR-mpls-te-oper:mpls-te/tunnels/summary</code> <code>Cisco-IOS-XR-ip-rsvp-oper:rsvp/interface-briefs/interface-brief</code> <code>Cisco-IOS-XR-mpls-te-oper:mpls-te/fast-reroute/protectations/protection</code> <code>Cisco-IOS-XR-mpls-te-oper:mpls-te/signalling-counters/signalling-summary</code> <code>Cisco-IOS-XR-mpls-te-oper:mpls-te/p2p-p2mp-tunnel/tunnel-heads/tunnel-head</code>
MPLS Label distribution protocol (MPLS-LDP)	<code>Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/nodes/node/bindings-summary-all</code> <code>Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/global/active/default-vrf/summary</code> <code>Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/nodes/node/default-vrf/neighbors/neighbor</code>
Routing	<code>Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/statistics-global</code> <code>Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/neighbors/neighbor</code> <code>Cisco-IOS-XR-ip-rib-ipv4-oper:rib/rib-table-ids/rib-table-id/summary-protos/summary-proto</code> <code>Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/levels/level/adjacencies/adjacency</code> <code>Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance/instance-active/default-vrf/process-info</code> <code>Cisco-IOS-XR-ip-rib-ipv6-oper:ipv6-rib/rib-table-ids/rib-table-id/summary-protos/summary-proto</code>



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 subscription 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 4](#)), or when an event occurs ([Event-driven Telemetry, on page 4](#)), 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 configuration 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 Daemon (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 parameters:

- **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.

$$\text{Cadence} = \text{Maximum} (\text{Global minimum-cadence defined}, (\text{Collection time for one collection} * \text{cadence-factor}))$$

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 tlv1-disable` command.

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 choose 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 13.

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 23.

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 gRPC in 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 2

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.

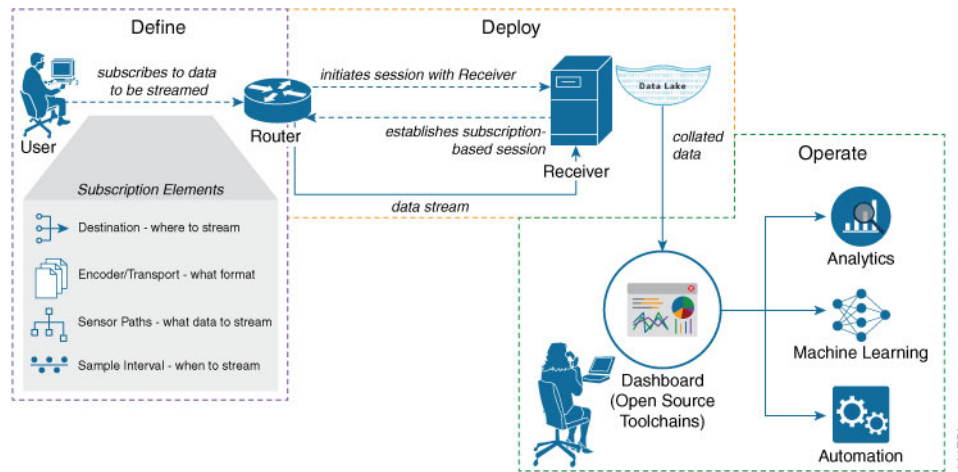


Note Watch this [video](#) to discover the power of real-time network management using model-driven telemetry.

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 14](#)

Monitor CPU Utilization Using Telemetry Data to Plan Network Infrastructure

The use case illustrates how, with the [Dial-Out Mode](#), you can use telemetry data to proactively monitor CPU utilization. Monitoring CPU utilization ensures efficient storage capabilities in your network. This use case describes the tools used in the open-sourced collection stack to store and analyse telemetry data.



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

Telemetry involves the following workflow:

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

Before you begin

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

Define a Subscription to Stream Data from Router to Receiver

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

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 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
- 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 name 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_mydmtrouter], [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

```

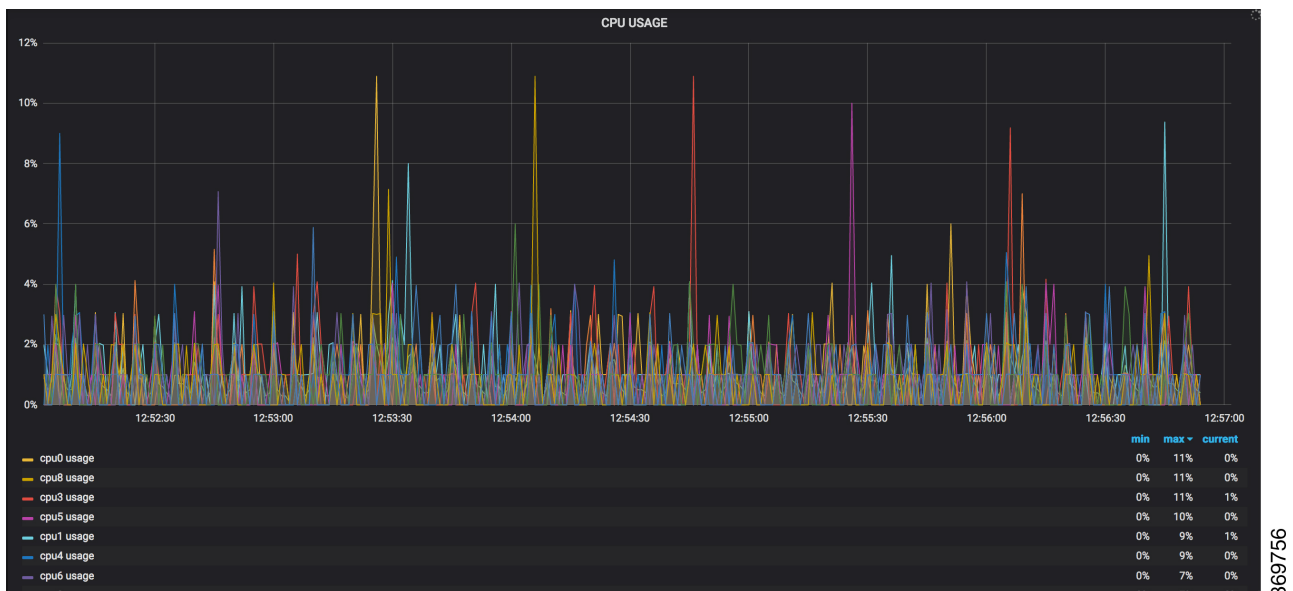
Step 3 Use Grafana to create a dashboard and visualize data about CPU usage.

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.

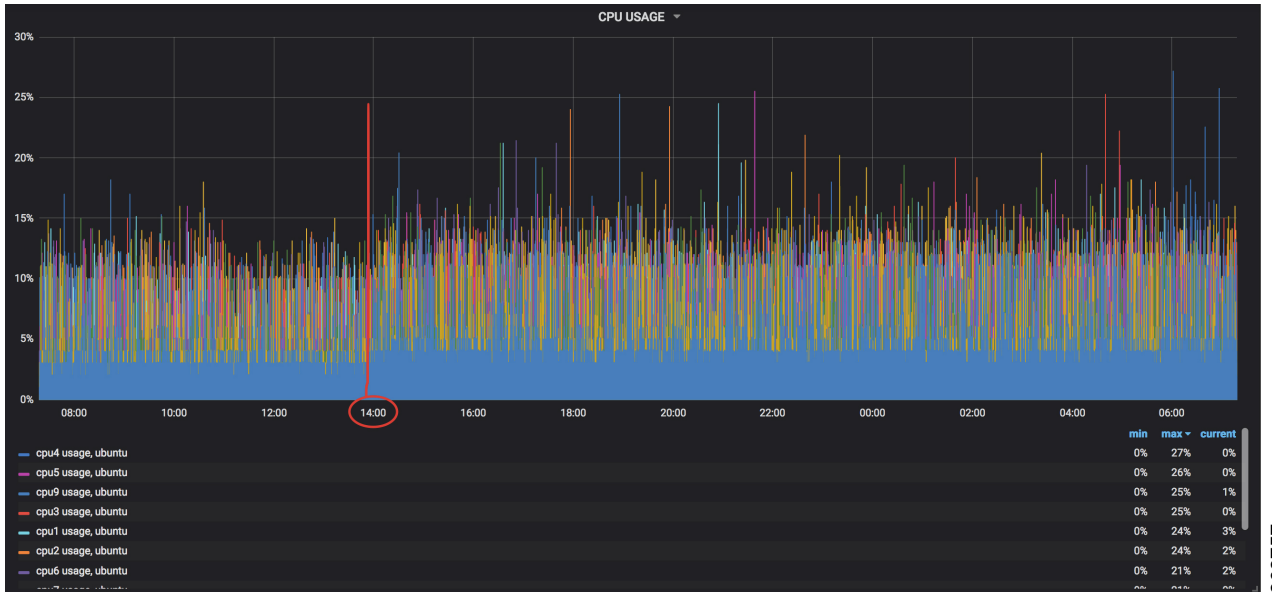
Figure 6: CPU Usage Graph with a Single Router



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

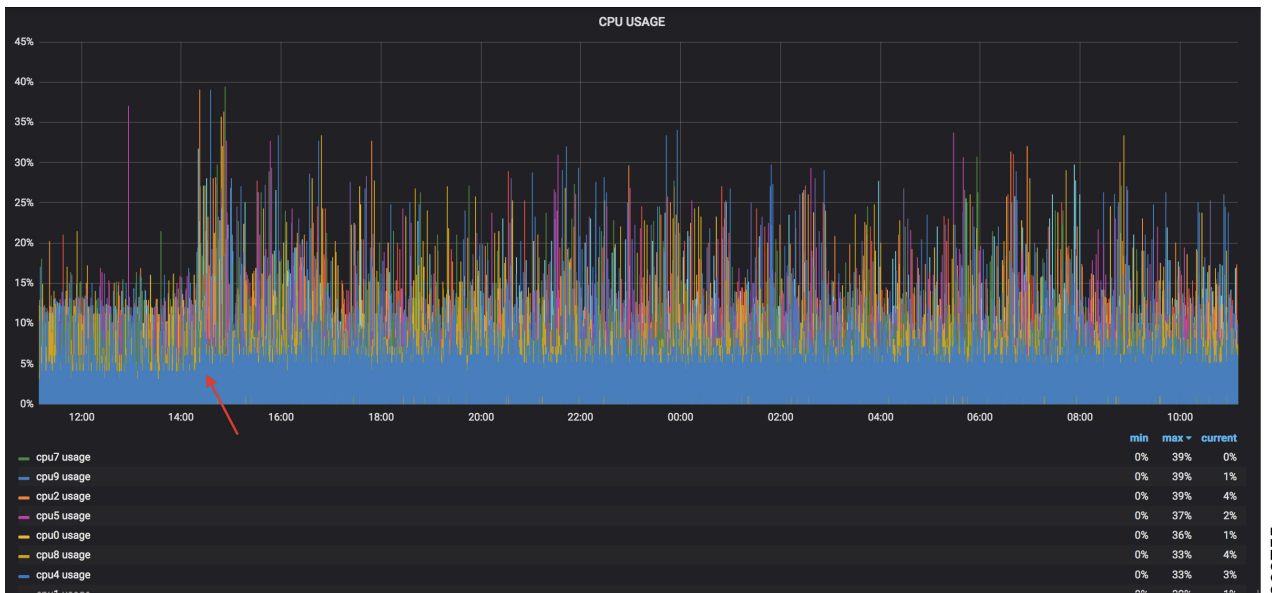


369757

Five routers

With five routers, the spikes peak upto approximately 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 3

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.

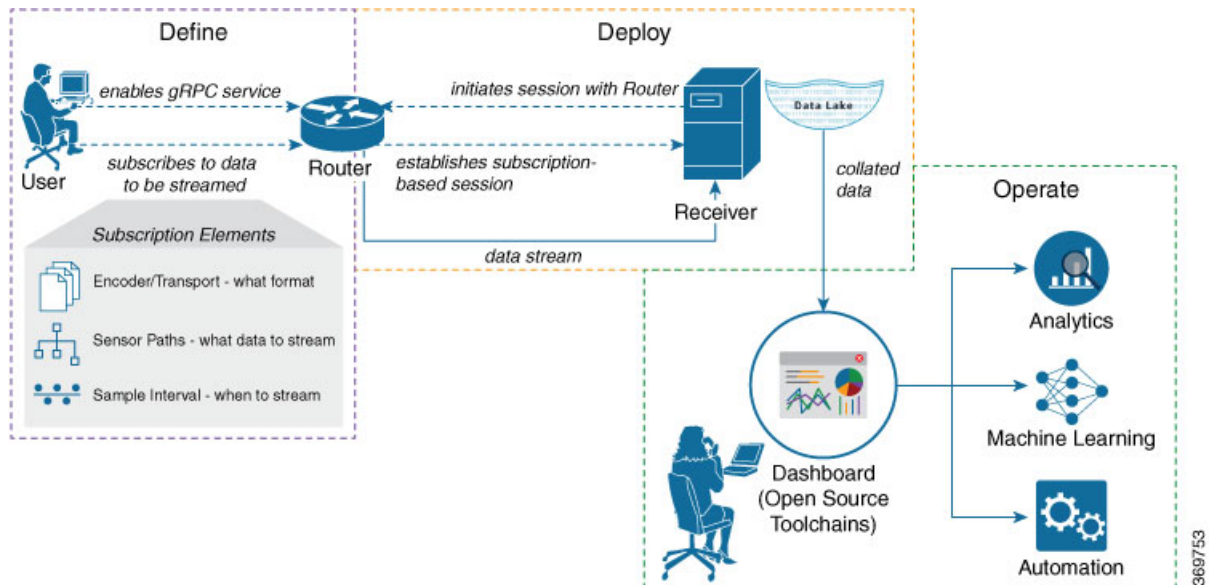


Note Watch this [video](#) to discover the power of real-time network management using model-driven telemetry.

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 24](#)

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 dependencies:

- 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](#). Separating 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  Vrf  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_myndtrouter], [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.

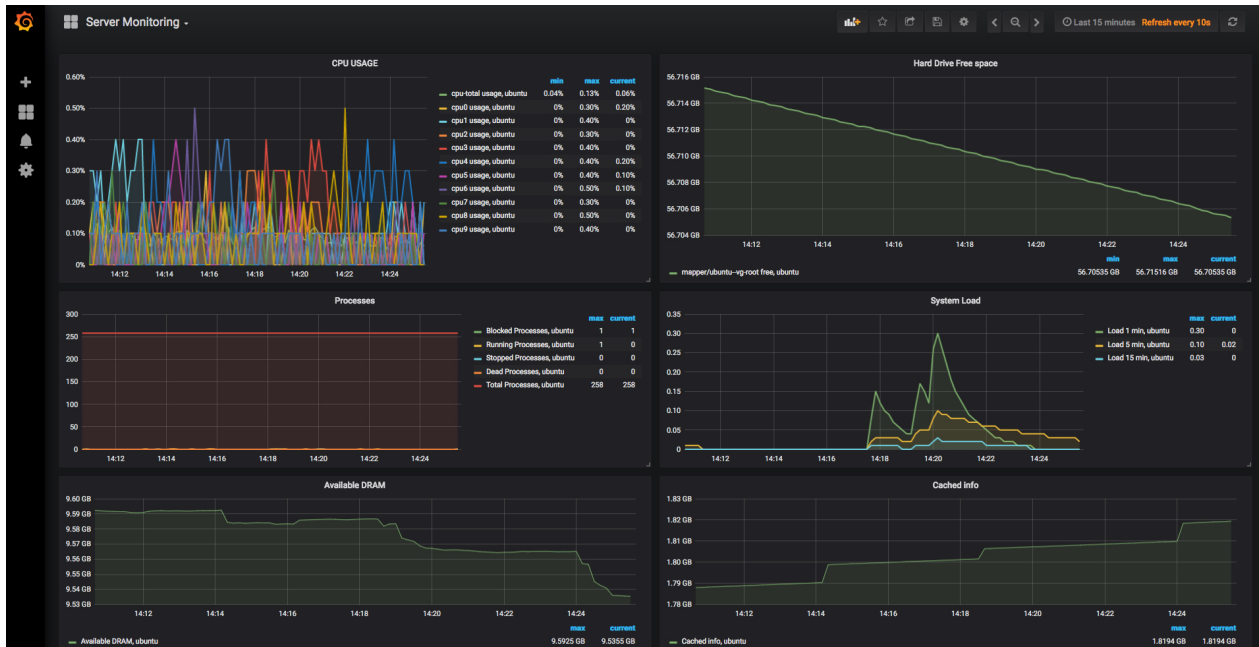
Step 2 Use Grafana to create a dashboard and visualize the streamed data.

Figure 10: Visual Analysis of Network Health using Telemetry Data



369765

Figure 11: Visual Analysis of System Monitoring using Telemetry Data



In conclusion, telemetry data shows that various parameters of the network can be monitored simultaneously. 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 4

Hardware Timestamp

Table 2: Feature History Table

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_1[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" }
```

- [Target-Defined Mode for Cached Generic Counters Data, on page 32](#)

Target-Defined Mode for Cached Generic Counters Data

Table 3: Feature History Table

Feature Name	Release Information	Description
Target-Defined Mode for Cached Generic Counters Data	Release 7.3.3	<p>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.</p> <p>This feature introduces support for the following sensor path:</p> <pre>Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters</pre>

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 grpl 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**

