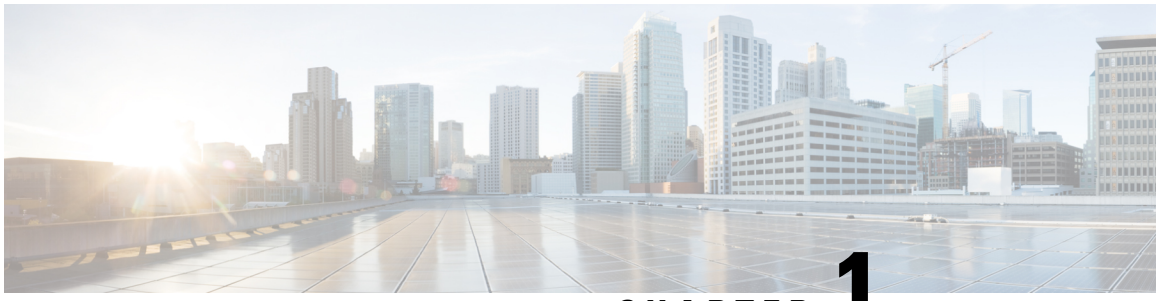
Telemetry Configuration Guide for Cisco NCS 560 Series Routers, IOS XR Release 7.1.x

First Published: 2020-01-29

Last Modified: 2020-01-28

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

CONTENTS



CHAPTER 1

New and Changed Feature Information

This section lists all the new and changed features for the *Telemetry Configuration Guide for Cisco NCS 560 Series Routers*.

New and Changed Telemetry Features

Feature	Description	Changed in Release	Where Documented
Support streaming telemetry data for ~~Cisco-IOS-XR-sysadmin-asic-errors-ael~~ Sysadmin model.	Support added to stream telemetry data for `Cisco-IOS-XR-sysadmin-asic-errors-ael` Sysadmin model.	Release 7.1.2	See Sensor Path, on page 13 topic for the list of supported Sysadmin data models. Obtain this data model from Github repository.

Feature	Description	Changed in Release	Where Documented
Support to poll specific processes to stream telemetry data.		Release 7.1.2	Obtain this data model from Github repository.

Feature	Description	Changed in Release	Where Documented

	Introduced `Cisco-IOS-XR-wdsysmon-fd-proc-oper.yang` data model with process keys to poll specific processes and stream telemetry data.
	NETCONF Request:
	`<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
	`<get>`
	`<filter>`
	`<process-monitoring xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-wdsysmon-fd-proc-oper">`
	`<nodes>`
	`<node>`
	`<node-name>0/RP0/CPU0</node-name>`
	`<process-name>`
	`<proc-cpu-utilizations>`
	`<proc-cpu-utilization>`
	`<process-name>`
	`dumper`
	`</process-name>`
	`</proc-cpu-utilization>`
	`</proc-cpu-utilizations>`
	`</process-name>`
	`</node>`
	`</nodes>`
	`</process-monitoring>`
	`</filter>`
	`</get>`
	`</rpc>`
	NETCONF Response:
	`<?xml version="1.0"?>`
	`<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">`
	`<data>`
	`<process-monitoring xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-wdsysmon-fd-proc-oper">`
	`<nodes>`
	`<node>`
	`<node-name>0/RP0/CPU0</node-name>`
	`<process-name>`
	`<proc-cpu-utilizations>`
	`<proc-cpu-utilization>`
	`<process-name>dumper</process-name>`
	`<total-cpu-one-minute>0</total-cpu-one-minute>`
	`<total-cpu-five-minute>0</total-cpu-five-minute>`
	`<total-cpu-fifteen-minute>0</total-cpu-fifteen-minute>`		

Feature	Description	Changed in Release	Where Documented
	<process-cpu> <process-name>dumper</process-name> <process-id>3572</process-id> <process-cpu-one-minute>0</process-cpu-one-minute> <process-cpu-five-minute>0</process-cpu-five-minute> <process-cpu-fifteen-minute>0</process-cpu-fifteen-minute> <thread-cpu> </thread-cpu> ------------------------ Truncated for brevity ---------------------------- </process-cpu> </proc-cpu-utilization> </proc-cpu-utilizations> </process-name> </node> </nodes> </process-monitoring> </data> </rpc-reply> The following example shows a telemetry query to fetch CPU utilization data in JSON format: mdt_exec -s Cisco-IOS-XR-wdsysmon-fd-proc-oper: process-monitoring/nodes/node[node-name=0/RP0/CPU0]/process-name/proc-cpu-utilizations/proc-cpu-utilization[process-name=bodls] -c 2000 -d output.json		

Feature	Description	Changed in Release	Where Documented

	The following stream of data shows the streamed data in JSON format:
	`Sub_id 200000001, flag 0, len 0`
	`Sub_id 200000001, flag 4, len 6496`
	`--------`
	{"node_id_str":"ios","subscription_id_str":"app_200000001", "encoding_path":"Cisco-IOS-XR-wdsysmon-fd-proc-oper: process-monitoring/nodes/node/process-name/proc-cpu-utilizations/ proc-cpu-utilization","collection_id":"4","collection_start_time": "1589478552400","msg_timestamp":"1589478552471", "data_json":[{"timestamp":"1589478552469","keys":[{"node-name": "0/RP0/CPU0"},{"process-name":"bcdls"}], "content":{"total-cpu-one-minute":0,"total-cpu-five-minute":1, "total-cpu-fifteen-minute":0, "process-cpu":[{"process-name":"bcdls","process-id":5113, "process-cpu-one-minute":0, "process-cpu-five-minute":0,"process-cpu-fifteen-minute":0, "thread-cpu":[{"thread-name": "lwm_service_thr","thread-id":5127,"process-cpu-one-minute":0, "process-cpu-five-minute":0, "process-cpu-fifteen-minute":0},{"thread-name":"qsm_service_thr", "thread-id":5128, "process-cpu-one-minute":0,"process-cpu-five-minute":0, "process-cpu-fifteen-minute":0}, {"thread-name":"bcdls","thread-id":5130,"process-cpu-one-minute":0, "process-cpu-five-minute":0, "process-cpu-fifteen-minute":0},{"thread-name":"bcdls","thread-id":5131, "process-cpu-one-minute":0, "process-cpu-five-minute":0,"process-cpu-fifteen-minute":0}, {"thread-name":"bcdls","thread-id":5132, "process-cpu-one-minute":0,"process-cpu-five-minute":0, "process-cpu-fifteen-minute":0},
	`----------Output trunacted for brevity ----------`		

Stream telemetry data using `openconfig-platform` data model	Streaming data related to the underlying characteristics of the device including the operational state or configuration of that device using openconfig-platform data model.	Release 7.1.1	Obtain this data model from Github repository.

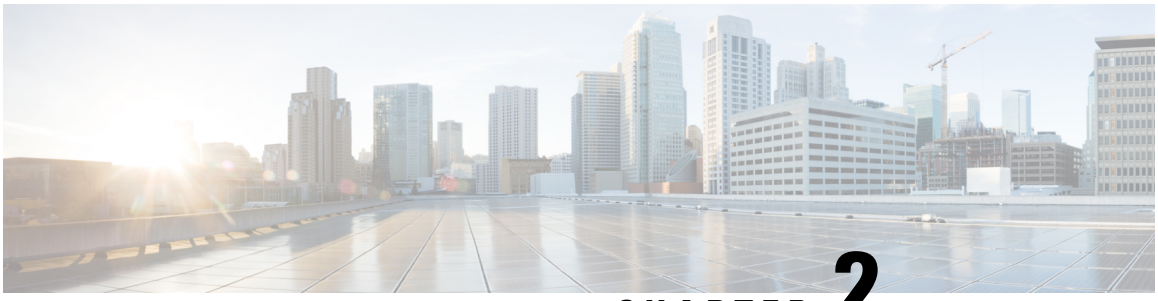
Feature	Description	Changed in Release	Where Documented

Congestion control for telemetry	Support to provide congestion management for telemetry.
	With congestion control, each destinationis allowed a maximum of 4000 outstanding messages. The events are throttled when the outstanding messages exceed 3000; throttling of cadence messages happen when outstanding messages exceed 250. Events have higher priority than cadence messages.
	A sample output is provided as follows:
	`Router#show telemetry model-driven destination`
	` DialIn_1002 1 192.x.x.x`
	` 19687 self-describing-gpb dialin Active`
	` TLS: False`
	`  Collection statistics:`
	`  Maximum tokens : 4000`
	`  Event tokens : 750`
	`  Cadence tokens : 723`
	`  Token processed at : <time-stamp>`
	`  Cadence token advertised at : <time-stamp>`
	`  Event token advertised at : >time-stamp>`
	`  GNMI initial synchronization time:`
	`  Pending queue size : 0`
	`  Processed events : 0`
	`  Collection tokens : 723`
		Release 7.1.1	NA

Feature	Description	Changed in Release	Where Documented

Support for retrieving information about CPU utilization at thread level	Enhanced `Cisco-IOS-XR-wdsysmon-fd-oper.yang` data model to include CPU utilization at thread level for each running process.
	The following example shows a sample output for a process:
	`<process-cpu>`
	`<process-name>tam_sync</process-name>`
	`<process-id>5062</process-id>`
	`<process-cpu-one-minute>0</process-cpu-one-minute>`
	`<process-cpu-five-minute>0</process-cpu-five-minute>`
	`<process-cpu-fifteen-minute>0</process-cpu-fifteen-minute>`
	`<thread-cpu>`
	`<thread-name>lwm_service_thr</thread-name>`
	`<thread-id>5063</thread-id>`
	`<process-cpu-one-minute>0</process-cpu-one-minute>`
	`<process-cpu-five-minute>0</process-cpu-five-minute>`
	`<process-cpu-fifteen-minute>0</process-cpu-fifteen-minute>`
	`</thread-cpu>`
	`</process-cpu>`
		Release 7.1.1	NA

Support for retreiving information about process threads	The `Cisco-IOS-XR-procthreadname-oper.yang` data model helps query thread-level details such as thread name, priority, state, stack size of a running processes.
	The following example shows a sample output:
	`<thread>`
	`<name>qsm_service_thr</name>`
	`<state>Sleeping</state>`
	`<stack>0K</stack>`
	`<pri>20</pri>`
	`<rtpri>0</rtpri>`
	`<jid>69381</jid>`
	`<tid>3866</tid>`
	`</thread>`
		Release 7.1.1	NA



CHAPTER 2

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?

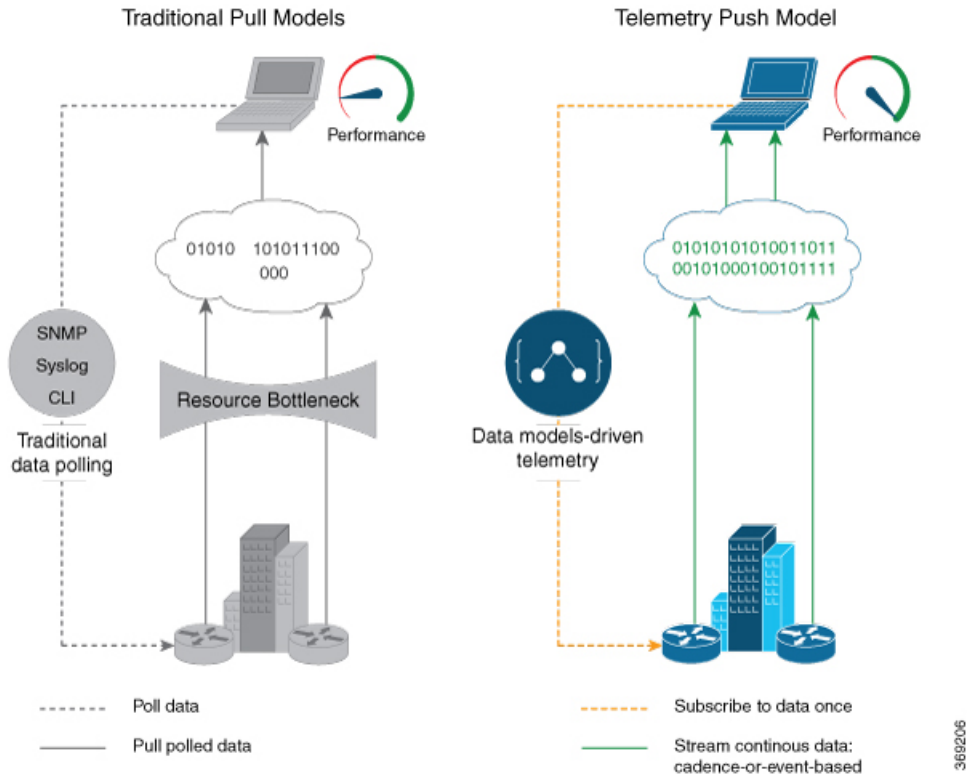
These traditional methods use a *pull* model to request information at regular intervals. The data that you collect may help you to efficiently monitor your network of a manageable size. However, as your network grows in complexity and scale, the data that you poll may be insufficient for efficient and 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 11
- Review Mechanisms to Stream Telemetry Data from a Router to a Destination, on page 11
- Learn About the Elements that Enable Streaming Telemetry Data , on page 13
- Explore the Methods to Establish a Telemetry Session, on page 17

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

Real-time telemetry data is useful in:

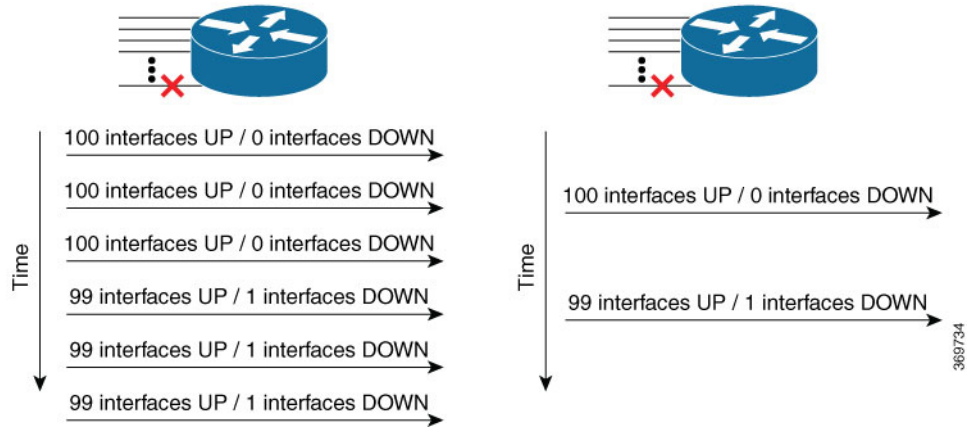
- **Managing network remotely:** The primary benefit of telemetry is the ability it offers you as an end user to monitor the state of a network element remotely. After the network is deployed, you cannot be physically present at the network site to find out what works, and what is cumbersome. With telemetry,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

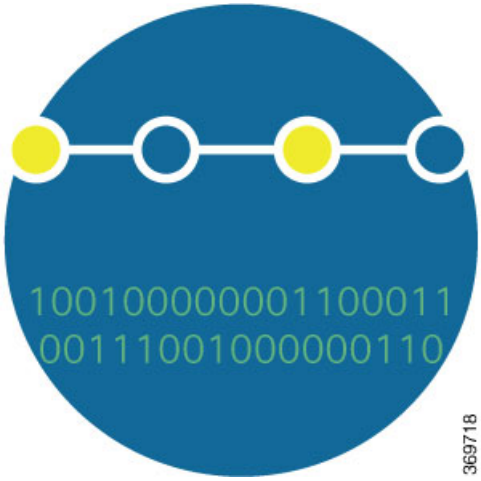


Cadence-driven Telemetry

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

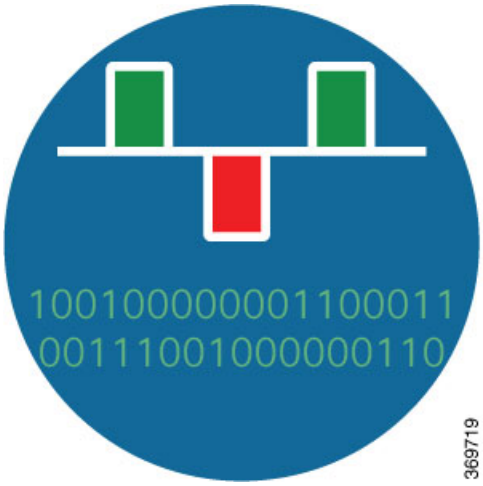


Event-driven Telemetry

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	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
Node summary	Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary
Forwarding information base (FIB)	Cisco-IOS-XR-fib-common-oper:fib-statistics/nodes/node/drops Cisco-IOS-XR-fib-common-oper:fib/nodes/node/protocols/protocol/vrfs/vrf/summary
MPLS Traffic engineering (MPLS-TE)	Cisco-IOS-XR-mpls-te-oper:mpls-te/tunnels/summary Cisco-IOS-XR-ip-rsvp-oper:rsvp/interface-briefs/interface-brief Cisco-IOS-XR-mpls-te-oper:mpls-te/fast-reroute/protections/protection Cisco-IOS-XR-mpls-te-oper:mpls-te/signalling-counters/signalling-summary Cisco-IOS-XR-mpls-te-oper:mpls-te/p2p-p2mp-tunnel/tunnel-heads/tunnel-head
MPLS Label distribution protocol (MPLS-LDP)	Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/nodes/node/bindings-summary-all Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/global/active/default-vrf/summary Cisco-IOS-XR-mpls-ldp-oper:mpls-ldp/nodes/node/default-vrf/neighbors/neighbor
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.

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 12), or when an event occurs (Event-driven Telemetry, on page 12), 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*.

gRPC Network Operations Interface

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

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

TLS Authentication

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

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

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.

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

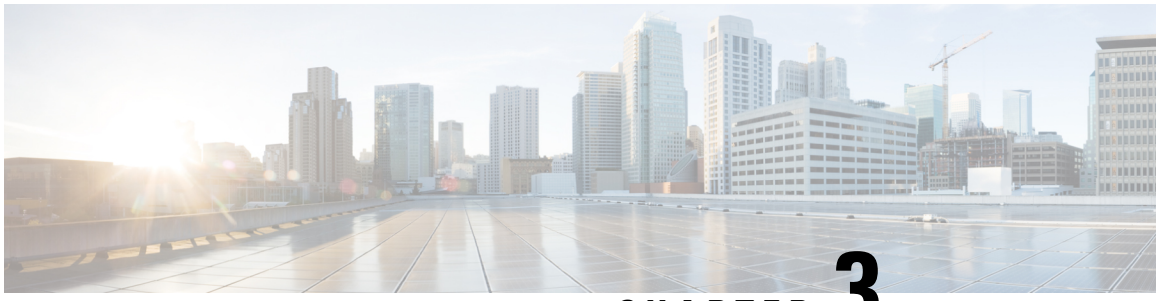
In a *dial-in* mode, a collector dials in to the router, and subscribes dynamically to one or more sensor paths specified in a subscription. The router is open for connections from the collector. This mode is useful to establish a single channel of communication with the router. Because the collector 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 3

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.

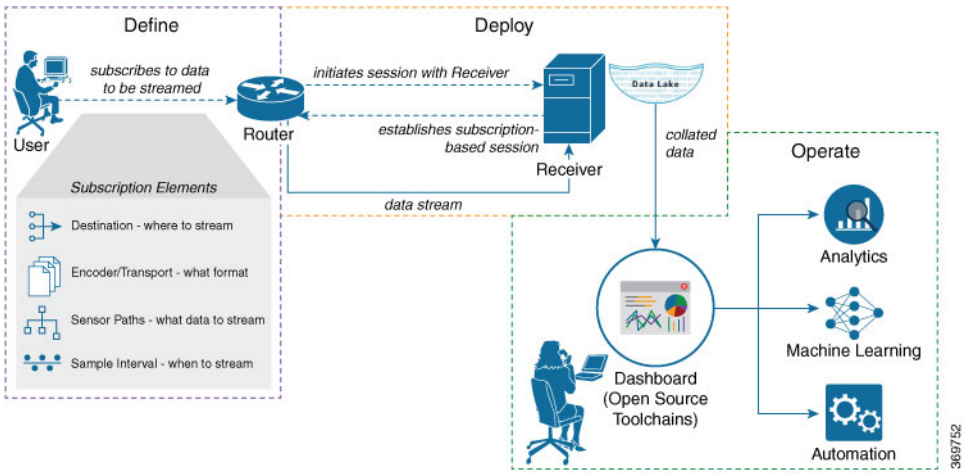
Telemetry works on a Subscription model where you subscribe to the data of interest in the form of Sensor Path. The sensor paths describes OpenConfig data models or native Cisco data models. You can access the OpenConfig and Native data models for telemetry from Github, a software development platform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

Procedure

Step 1 Create one or more destinations to collect telemetry data from a router. Define a destination-group to contain the details about the destinations. Include the destination address (ipv4 or ipv6), 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
- `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.

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.

Procedure

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

Example:

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

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

Example:

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

Subscription:  CPU-Utilization
-------------
  State:       NA
  Source Interface:       GigabitEthernet0_0_0_0( 0x0)
  Sensor groups:
  Id: Monitor-CPU
    Sample Interval:      30000 ms
    Sensor Path:          Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
    Sensor Path State:    Resolved

  Destination Groups:
  Group Id: CPU-Health
    Destination IP:       172.0.0.0
    Destination Port:     57500
    Encoding:             self-describing-gpb
```

```
  Transport:            tcp
  State:                NA
  No TLS

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

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

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.

Procedure

Step 1 Start Pipeline, and enter your router credentials.

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

Example:

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

Startup pipeline
```

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

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

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

Example:

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

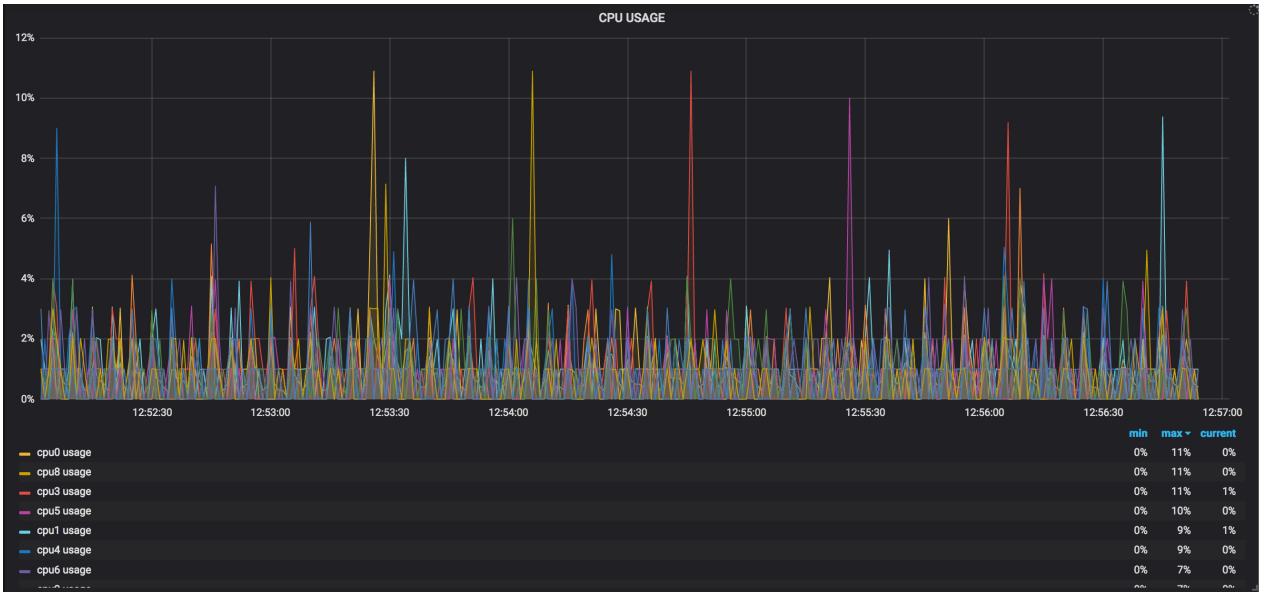
Step 3 Use Grafana to create a dashboard and visualize data about CPU usage.

One router

The router pushes the counters every five seconds.

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

Figure 6: CPU Usage Graph with a Single Router

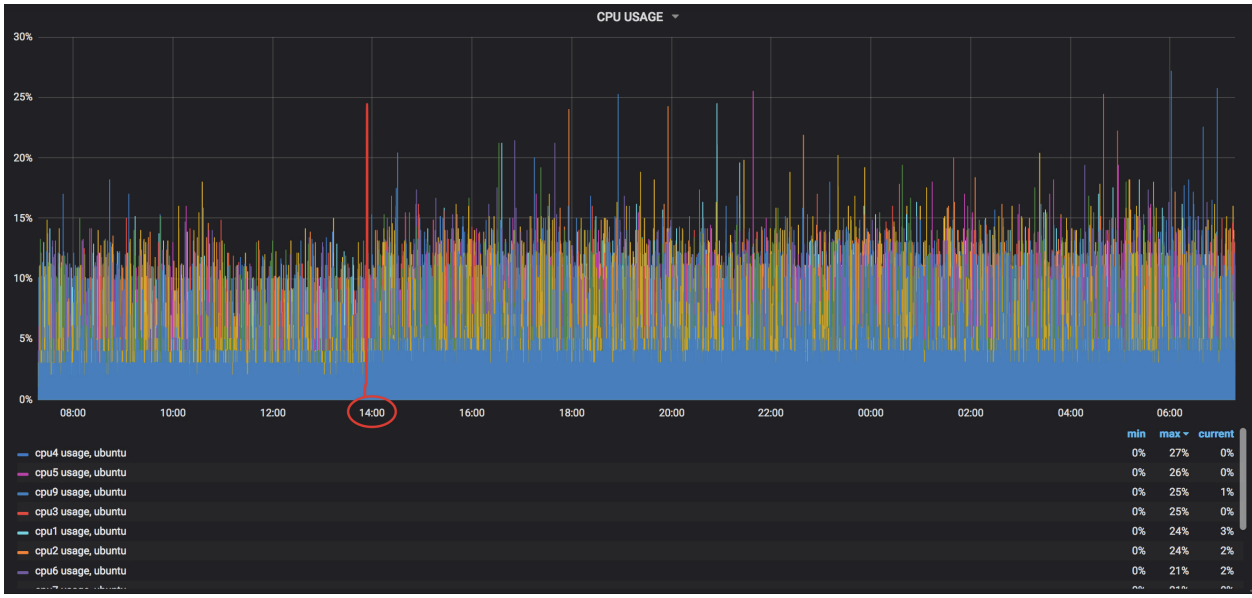


369756

Two routers

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

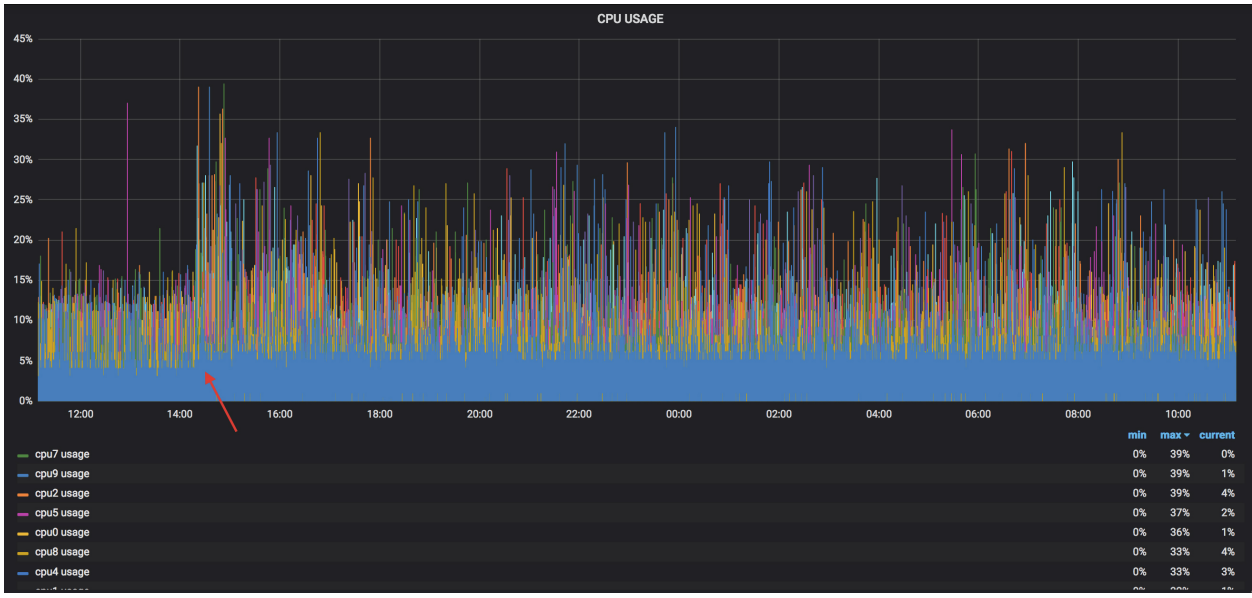
Figure 7: CPU Usage Graph with Two Routers



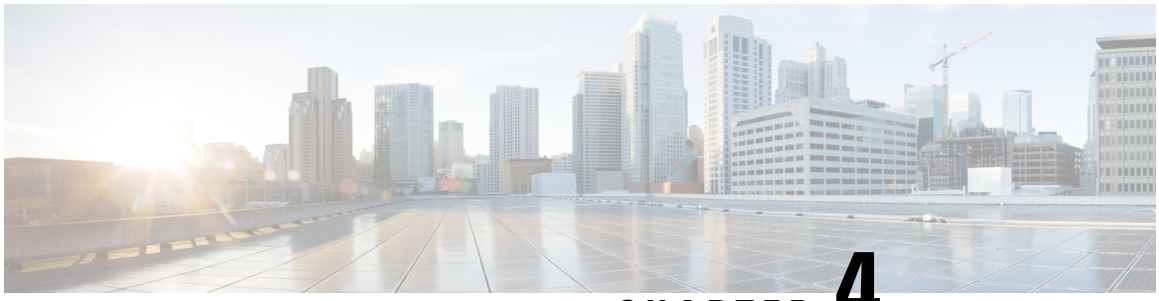
Five routers

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



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 4

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.

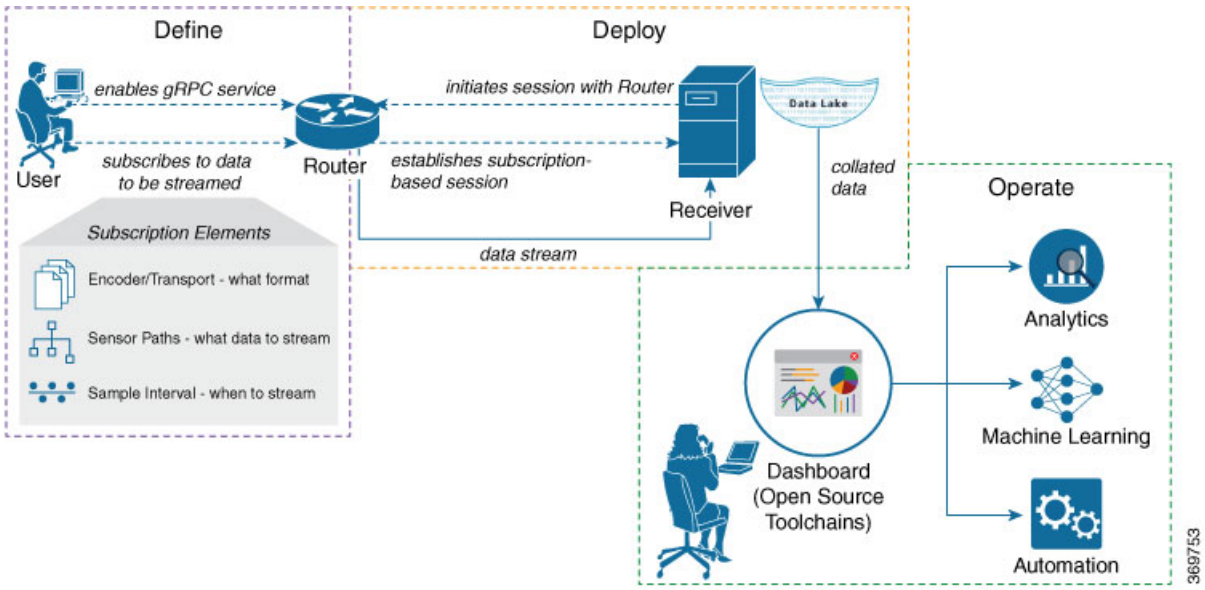
Telemetry works on a Subscription model where you subscribe to the data of interest in the form of Sensor Path. The sensor paths describes OpenConfig data models or native Cisco data models. You can access the OpenConfig and Native data models for telemetry from Github, a software development platform 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 : 57300
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.

Procedure

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

Example:

```
sensor-group health
  sensor-path Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-utilization
  sensor-path Cisco-IOS-XR-nto-misc-oper:memory-summary/nodes/node/summary
  sensor-path Cisco-IOS-XR-shellutil-oper:system-time/uptime
 !
 sensor-group interfaces
  sensor-path
Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters

  sensor-path Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-summary
 !
 sensor-group optics
  sensor-path
Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info
 !
 sensor-group routing
  sensor-path
Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/levels/level/adjacencies/adjacency
  sensor-path Cisco-IOS-XR-clns-isis-oper:isis/instances/instance/statistics-global
  sensor-path
Cisco-IOS-XR-ip-rib-ipv4-oper:rib/vrfs/vrf/afs/af/safs/saf/ip-rib-route-table-names/ip-rib-route-table-name/protocol/isis/as/information

  sensor-path
Cisco-IOS-XR-ipv4-bgp-oper:bgp/instances/instance/instance-active/default-vrf/process-info
 !
 sensor-group mpls-te
 sensor-path Cisco-IOS-XR-mpls-te-oper:mpls-te/tunnels/summary
 sensor-path Cisco-IOS-XR-ip-rsvp-oper:rsvp/interface-briefs/interface-brief
 sensor-path Cisco-IOS-XR-ip-rsvp-oper:rsvp/counters/interface-messages/interface-message
 !
```

Step 2 Subscribe to telemetry data that is streamed from a router. A Subscription binds the sensor-group, and sets the streaming method. The streaming method can be Cadence-driven 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.

Procedure

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

Example:

```
Router#show telemetry model-driven subscription
Thu Jan  16 09:48:14.293 UTC
Subscription:  health                  State: Active
-------------
  Sensor groups:
  Id                              Interval(ms)        State
  health                          30000               Resolved

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

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

Subscription:  routing                 State: NA
```

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

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

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

  Destination Groups:
  Id                 Encoding            Transport   State   Port    Vrf     IP

  CPU-Health         self-describing-gpb tcp         NA      57500           172.0.0.0
    No TLS
```

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

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

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

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

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

Consider that the router is monitored for the following parameters:

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

- RSVP control messages and bandwidth allocation for each interface

Procedure

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

Example:

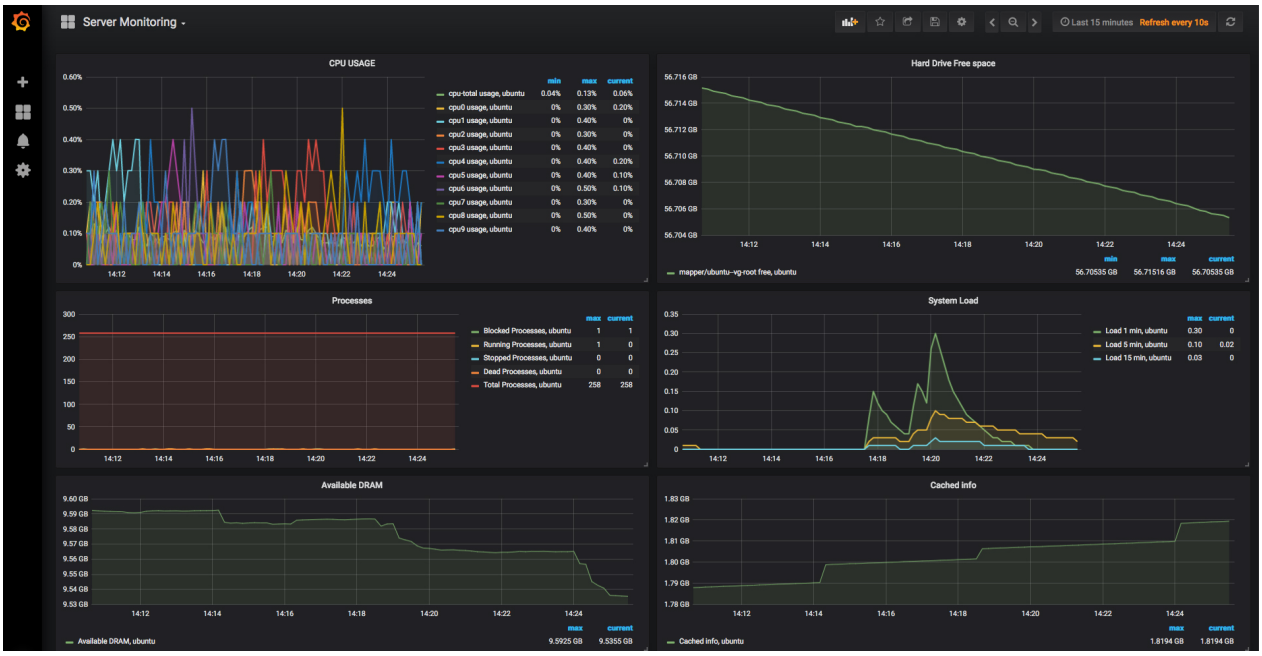
```
$ bin/pipeline -config pipeline.conf

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

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

The streamed telemetry data is stored in InfluxDB.

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

Figure 10: Visual Analysis of Network Health using Telemetry Data

Figure 11: Visual Analysis of System Monitoring using Telemetry Data



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.