Model-Driven Telemetry

This chapter contains the following topics:

About Telemetry

Collecting data for analyzing and troubleshooting has always been an important aspect in monitoring the health of a network.

Cisco NX-OS provides several mechanisms such as SNMP, CLI, and Syslog to collect data from a network. These mechanisms have limitations that restrict automation and scale. One limitation is the use of the pull model, where the initial request for data from network elements originates from the client. The pull model does not scale when there is more than one network management station (NMS) in the network. With this model, the server sends data only when clients request it. To initiate such requests, continual manual intervention is required. This continual manual intervention makes the pull model inefficient.

A push model continuously streams data out of the network and notifies the client. Telemetry enables the push model, which provides near-real-time access to monitoring data.

Telemetry Components and Process

Telemetry consists of four key elements:

- **Data Collection**—Telemetry data is collected from the Data Management Engine (DME) database in branches of the object model specified using distinguished name (DN) paths. The data can be retrieved periodically (frequency-based) or only when a change occurs in any object on a specified path (event-based). You can use the NX-API to collect frequency-based data.
- **Data Encoding**—The telemetry encoder encapsulates the collected data into the desired format for transporting.

  NX-OS encodes telemetry data in the Google Protocol Buffers (GPB) and JSON format.
- **Data Transport**—NX-OS transports telemetry data using HTTP for JSON encoding and the Google remote procedure call (gRPC) protocol for GPB encoding. The gRPC receiver supports message sizes greater than 4 MB. (Telemetry data using HTTPS is also supported if a certificate is configured.)

UDP and secure UDP (DTLS) are supported as telemetry transport protocols. You can add destinations that receive UDP. The encoding for UDP and secure UDP can be GPB or JSON.

Telemetry supports streaming to IPv6 destinations and IPv4 destinations.

Use the following command to configure the UDP transport to stream data using a datagram socket either in JSON or GPB:

```
destination-group num
  ip address xxx.xxx.xxx.xxx port xxxx protocol UDP encoding {JSON | GPB }
```

Example for an IPv4 destination:

```
destination-group 100
  ip address 171.70.55.69 port 50001 protocol UDP encoding GPB
```

Example for an IPv6 destination:

```
destination-group 100
  ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB
```

The UDP telemetry is with the following header:

```
typedef enum tm_encode_ {
  TM_ENCODE_DUMMY,
  TM_ENCODE_GPB,
  TM_ENCODE_JSON,
  TM_ENCODE_XML,
  TM_ENCODE_MAX,
} tm_encode_type_t;

typedef struct tm_pak_hdr_ {
  uint8_t version; /* 1 */
  uint8_t encoding;
  uint16_t msg_size;
  uint8_t secure;
  uint8_t padding;
}__attribute__ ((packed, aligned (1))) tm_pak_hdr_t;
```

Use the first 6 bytes in the payload to process telemetry data using UDP, using one of the following methods:

- Read the information in the header to determine which decoder to use to decode the data, JSON or GPB, if the receiver is meant to receive different types of data from multiple endpoints.
- Remove the header if you are expecting one decoder (JSON or GPB) but not the other.

Note Depending on the receiving operation system and the network load, using the UDP protocol may result in packet drops.

- **Telemetry Receiver**—A telemetry receiver is a remote management system or application that stores the telemetry data.

The GPB encoder stores data in a generic key-value format. The encoder requires metadata in the form of a compiled `.proto` file to translate the data into GPB format.

In order to receive and decode the data stream correctly, the receiver requires the `.proto` file that describes the encoding and the transport services. The encoding decodes the binary stream into a key value string pair.

A telemetry `.proto` file that describes the GPB encoding and gRPC transport is available on Cisco's GitLab: https://github.com/CiscoDevNet/nx-telemetry-proto

High Availability of the Telemetry Process

High availability of the telemetry process has the following behaviors:

- **System Reload**—During a system reload, any telemetry configuration, and streaming services are restored.
- **Supervisor Failover**—Although telemetry is not on hot standby, telemetry configuration, and streaming services are restored when the new active supervisor is running.
- **Process Restart**—If the telemetry process freezes or restarts for any reason, configuration and streaming services are restored when telemetry restarts.

Licensing Requirements for Telemetry

Product	License Requirement
Cisco NX-OS	Telemetry requires no license. Any feature that is not included in a license package is bundled with the Cisco NX-OS image and is provided at no extra charge to you. For a complete explanation of the Cisco NX-OS licensing scheme, see the *Cisco NX-OS Licensing Guide*.

Installing and Upgrading Telemetry

Installing the Application

The telemetry application is packaged as a feature RPM and included with the NX-OS release. The RPM is installed by default as part of the image bootup. After installation, you can start the application using the **feature telemetry** command. The RPM file is in the `/rpms` directory and has the following name:

telemetry-*version-build_ID***.libn32_n3000.rpm**

As in the following example:

```
telemetry-2.0.0.lib32_n3000.rpm
```

Installing Incremental Updates and Fixes

Copy the RPM to the device bootflash and use the following commands from the `bash` prompt:

```
feature bash
run bash sudo su
```

Then copy the RPM to the device bootflash. Use the following commands from the `bash` prompt:

yum upgrade *telemetry_new_version*.**rpm**

When the application restarts, it is upgraded and the change appears.

Downgrading to a Previous Version

To downgrade the telemetry application to a previous version, use the following command from the `bash` prompt:

```
yum downgrade telemetry
```

Verifying the Active Version

To verify the active version, run the following command from the switch `exec` prompt:

```
show install active
```

Note The `show install active` command shows the active installed RPM only after an upgrade has occurred. The default RPM that comes bundled with the NX-OS is not displayed.

Guidelines and Limitations

Telemetry has the following configuration guidelines and limitations:

- Telemetry is supported in Cisco NX-OS releases that support the data management engine (DME) Native Model.
- Telemetry supports DME data collection, NX-API data sources, Google protocol buffer (GPB) encoding over Google Remote Procedure Call (gRPC) transport, and JSON encoding over HTTP.
- The smallest sending interval (cadence) supported is five seconds for a depth of 0. The minimum cadence values for depth values greater than 0 depends on the size of the data being streamed out. Configuring cadences below the minimum value may result in undesirable system behavior.
- Up to five remote management receivers (destinations) are supported. Configuring more than five remote receivers may result in undesirable system behavior.
- If a telemetry receiver goes down, other receivers see data flow interrupted. The failed receiver must be restarted. Then start a new connection with the switch by unconfiguring then reconfiguring the failed receiver's IP address under the destination group.
- Telemetry can consume up to 20% of the CPU resource.
- To configure SSL certificate-based authentication and the encryption of streamed data, you can provide a self-signed SSL certificate with **certificate** *ssl cert path* **hostname "CN"** command.

Configuration Commands After Downgrading to an Older Release

After a downgrade to an older release, some configuration commands or command options can fail because the older release may not support them. As a best practice when downgrading to an older release, unconfigure and reconfigure the telemetry feature after the new image comes up. By doing so, you avoid possible failure of unsupported commands or command options.

The following example shows this procedure:

- Copy the telemetry configuration to a file:

```
switch# show running-config | section telemetry
```

```
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch# show running-config | section telemetry > telemetry_running_config
switch# show file bootflash:telemetry_running_config
feature telemetry
telemetry
  destination-group 100
    ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
    use-chunking size 4096
  sensor-group 100
    path sys/bgp/inst/dom-default depth 0
  subscription 600
    dst-grp 100
    snsr-grp 100 sample-interval 7000
switch#
```

- Execute the downgrade operation. When the image comes up and the switch is ready, copy the telemetry configurations back to the switch:

```
switch# copy telemetry_running_config running-config echo-commands
`switch# config terminal`
`switch(config)# feature telemetry`
`switch(config)# telemetry`
`switch(config-telemetry)# destination-group 100`
`switch(conf-tm-dest)# ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB `
`switch(conf-tm-dest)# sensor-group 100`
`switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0`
`switch(conf-tm-sensor)# subscription 600`
`switch(conf-tm-sub)# dst-grp 100`
`switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000`
`switch(conf-tm-sub)# end`
Copy complete, now saving to disk (please wait)...
Copy complete.
switch#
```

gRPC Error Behavior

The switch client disable the connection to the gRPC receiver if the gRPC receiver sends 20 errors. You will need to unconfigure then reconfigure the receiver's IP address under the destination group to enable the gRPC receiver. Errors include:

- The gRPC client sends the wrong certificate for secure connections.
- The gRPC receiver takes too long to handle client messages and incurs a timeout. Avoid timeouts by processing messages using a separate message processing thread.

Telemetry Compression for gRPC Transport

Telemetry compression support is available for gRPC transport. You can use the **use-compression gzip** command to enable compression. (Disable compression with the **no use-compression gzip** command.)

The following example enables compression:

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(config-tm-dest-profile)# use-compression gzip
```

The following example shows that compression is enabled:

```
switch(conf-tm-dest)# show telemetry transport 0 stats

Session Id:                     0
Connection Stats
   Connection Count             0
   Last Connected:              Never
   Disconnect Count             0
   Last Disconnected:           Never
Transmission Stats
   Compression:                 gzip
   Source Interface:            loopback1(1.1.3.4)
   Transmit Count:              0
   Last TX time:                None
   Min Tx Time:                 0                    ms
   Max Tx Time:                 0                    ms
   Avg Tx Time:                 0                    ms
   Cur Tx Time:                 0                    ms


switch2(config-if)# show telemetry transport 0 stats

Session Id: 0
Connection Stats
Connection Count 0
Last Connected: Never
Disconnect Count 0
Last Disconnected: Never
Transmission Stats
Compression: disabled
Source Interface: loopback1(1.1.3.4)
Transmit Count: 0
Last TX time: None
Min Tx Time: 0 ms
Max Tx Time: 0 ms
Avg Tx Time: 0 ms
Cur Tx Time: 0 ms
switch2(config-if)#
```

The following is an example of use-compression as a POST payload:

```
{
              "telemetryDestProfile": {
                "attributes": {
                  "adminSt": "enabled"
                },
                "children": [
                  {
                    "telemetryDestOptCompression": {
                      "attributes": {
                        "name": "gzip"
                      }
                    }
                  }
                ]
              }
            }
```

Support for gRPC Chunking

For streaming to occur successfully, you must enable chunking if gRPC has to send an amount of data greater than 12 MB to the receiver.

gRPC chunking must be done by the gRPC user. Fragmentation occurs on the gRPC client side and reassembly occurs on the gRPC server side. Telemetry is still bound to memory and data can be dropped if the memory size is more than the allowed limit of 12 MB for telemetry. In order to support chunking, use the telemetry `.proto` file that is available at Cisco's GibLab, which has been updated for gRPC chunking, as described in Telemetry Components and Process, on page 1.

The chunking size is from 64 through 4096 bytes.

Following shows a configuration example through the NX-API CLI:

```
feature telemetry
!
telemetry
  destination-group 1
    ip address 171.68.197.40 port 50051 protocol gRPC encoding GPB
    use-chunking size 4096
  destination-group 2
    ip address 10.155.0.15 port 50001 protocol gRPC encoding GPB
    use-chunking size 64
  sensor-group 1
    path sys/intf depth unbounded
  sensor-group 2
    path sys/intf depth unbounded
  subscription 1
    dst-grp 1
    snsr-grp 1 sample-interval 10000
  subscription 2
    dst-grp 2
    snsr-grp 2 sample-interval 15000
```

Following shows a configuration example through the NX-API REST:

```
{
    "telemetryDestGrpOptChunking": {
        "attributes": {
            "chunkSize": "2048",
            "dn": "sys/tm/dest-1/chunking"
        }
    }
}
```

The following error message appears on systems that do not support gRPC chunking:

```
switch-1(conf-tm-dest)# use-chunking size 200
ERROR: Operation failed: [chunking support not available]
```

NX-API Sensor Path Limitations

NX-API can collect and stream switch information not yet in the DME using **show** commands. However, using the NX-API instead of streaming data from the DME has inherent scale limitations as outlined:

- The switch backend dynamically processes NX-API calls such as **show** commands,
- NX-API spawns several processes that can consume up to a maximum of 20% of the CPU.
- NX-API data translates from the CLI to XML to JSON.

The following is a suggested user flow to help limit excessive NX-API sensor path bandwidth consumption:

1. Check whether the **show** command has NX-API support. You can confirm whether NX-API supports the command from the VSH with the pipe option: `show <command> | json` or `show <command> | json pretty`.

   Note Avoid commands that take the switch more than 30 seconds to return JSON output.

2. Refine the **show** command to include any filters or options.
   - Avoid enumerating the same command for individual outputs; for example, **show vlan id 100** , **show vlan id 101** , and so on. Instead, use the CLI range options; for example, **show vlan id 100-110,204** , whenever possible to improve performance.

     If you need only the summary or counter, avoid dumping a whole show command output. By doing so, you limit the bandwidth and data storage that is required for data collection.

3. Configure telemetry with sensor groups that use NX-API as their data sources. Add the **show** commands as sensor paths
4. Configure telemetry with a cadence of five times the processing time of the respective **show** command to limit CPI usage.
5. Receive and process the streamed NX-API output as part of the existing DME collection.

Telemetry VRF Support

Telemetry VRF support allows you to specify a transport VRF, which means that the telemetry data stream can egress through front-panel ports and avoid possible competition between SSH/NGINX control sessions.

You can use the **use-vrf** *vrf-name* command to specify the transport VRF.

The following example specifies the transport VRF:

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(config-tm-dest-profile)# use-vrf test_vrf
```

The following is an example of use-vrf as a POST payload:

```
{
               "telemetryDestProfile": {
                 "attributes": {
                   "adminSt": "enabled"
                 },
                 "children": [
                   {
                     "telemetryDestOptVrf": {
                       "attributes": {
                         "name": "default"
                       }
                     }
                   }
                 ]
```

```
    }
}
```

Support for Streaming of YANG Models

The YANG ("Yet Another Next Generation") data modeling language is supported as part of telemetry. Both device YANG and open config YANG model data streaming are supported.

Configuring Telemetry Using the CLI

Configuring Telemetry Using the NX-OS CLI

The following steps enable streaming telemetry and configuring the source and destination of the data stream. These steps also include optional steps to enable and configure SSL/TLS certificates and GPB encoding.

Procedure

	Command or Action	Purpose

Step 1	(Optional) **openssl** *argument*
	Example:
	Generate an SSL/TLS certificate using a specific argument, such as the following:
	• To generate a private RSA key: **openssl genrsa** *-cipher* **-out** *filename.key cipher-bit-length*
	For example:
	`switch# openssl genrsa -des3 -out server.key 2048`
	• To write the RSA key: **openssl rsa -in** *filename.key* **-out** *filename.key*
	For example:
	`switch# openssl rsa -in server.key -out server.key`
	• To create a certificate that contains the public or private key: **openssl req**
	-encoding-standard **-new** **-new** *filename.key* **-out** *filename.csr* **-subj '/CN=localhost'**
	For example:
	`switch# openssl req -sha256 -new -key server.key -out server.csr -subj '/CN=localhost'`
		Create an SSL or TLS certificate on the server that will receive the data, where *private*.key file is the private key and the *public*.crt is the public key.

	Command or Action	Purpose
	• To create a public key: **openssl x509 -req** *-encoding-standard* **-days** *timeframe* **-in** *filename.csr* **-signkey** *filename.key* **-out** *filename.csr* For example: `switch# openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt`	

Step 2	**configure terminal**
	Example:
	`switch# configure terminal`
	`switch(config)#`
		Enter the global configuration mode.

Step 3	**feature telemetry**	Enable the streaming telemetry feature.
Step 4	**feature nxapi**	Enable NX-API.
Step 5	**nxapi use-vrf management**	Enable the VRF management to be used for NX-API communication.

Step 6	**telemetry**
	Example:
	`switch(config)# telemetry`
	`switch(config-telemetry)#`
		Enter configuration mode for streaming telemetry.

Step 7	(Optional) **certificate** *certificate_path host_URL* **Example:** `switch(config-telemetry)# certificate /bootflash/server.key localhost`	Use an existing SSL/TLS certificate.

Step 8	(Optional) Specify a transport VRF or enable telemetry compression for gRPC transport.
	Example:
	`switch(config-telemetry)# destination-profile`
	`switch(conf-tm-dest-profile)# use-vrf default`
	`switch(conf-tm-dest-profile)# use-compression gzip`
	`switch(conf-tm-dest-profile)# use-retry size 10`
	`switch(conf-tm-dest-profile)# source-interface loopback1`
		• Enter the **destination-profile** command to specify the default destination profile.
		• Enter any of the following commands:
		  • **use-vrf** *vrf* to specify the destination VRF.
		  • **use-compression gzip** to specify the destination compression method.
		  • **use-retry size** *size* to specify the send retry details, with a retry buffer size from 10 through 1500 megabytes.
		  • **source-interface** *interface-name* to stream data from the configured

	Command or Action	Purpose
		interface to a destination with the source IP address. **Note** After configuring the **use-vrf** command, you must configure a new destination IP address within the new VRF. However, you may re-use the same destination IP address by unconfiguring and reconfiguring the destination. This ensures that the telemetry data streams to the same destination IP address in the new VRF.

Step 9	**sensor-group** *sgrp_id*
	Example:
	`switch(config-telemetry)# sensor-group 100`
	`switch(conf-tm-sensor)#`
		Create a sensor group with ID *srgp_id* and enter sensor group configuration mode.
		Currently only numeric ID values are supported. The sensor group defines nodes that will be monitored for telemetry reporting.

Step 10	(Optional) **data-source** *data-source-type* **Example:** `switch(config-telemetry)# data-source NX-API`	Select a data source. Select from either YANG, DME or NX-API as the data source. **Note** DME is the default data source.

Step 11	**path** *sensor_path* **depth 0** [**filter-condition** *filter*]
	Example:
	• The following command is applicable for DME, not for NX-API or YANG:
	`switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition eq(l2BD.operSt, "down")`
	Use the following syntax for state-based filtering to trigger only when **operSt** changes from **up** to **down**, with no notifications of when the MO changes.
	`switch(conf-tm-sensor)# path sys/bd/bd-[vlan-100] depth 0 filter-condition and(updated(l2BD.operSt),eq(l2BD.operSt,"down"))`
	• The following command is applicable for NX-API, not for DME or YANG:
	`switch(conf-tm-sensor)# path "show interface" depth 0`
		Add a sensor path to the sensor group.
		• The **depth** setting specifies the retrieval level for the sensor path. Depth settings of **0** - **32**, **unbounded** are supported.
		Note **depth 0** is the default depth.
		NX-API-based sensor paths can only use **depth 0**.
		If a path is subscribed for the event collection, the depth only supports 0 and unbounded. Other values are treated as 0.
		• The optional **filter-condition** parameter can be specified to create a specific filter for event-based subscriptions.
		For state-based filtering, the filter returns both when a state has changed and when an event has occurred during the specified state. That is, a filter condition for the DN **sys/bd/bd-[vlan]** of **eq(l2Bd.operSt,**

	Command or Action	Purpose

	• The following command is applicable for device YANG:
	`switch(conf-tm-sensor)# path Cisco-NX-OS-device:System/bgp-items/inst-items`
	• The following command is applicable for OpenConfig YANG:
	`switch(conf-tm-sensor)# path openconfig-bgp:bgp`
		"down") triggers when the operSt changes, and when the DN's property changes while the operSt remains **down**. One example is when a **no shutdown** command is issued while the VLAN is operationally **down**.
		Note query-condition parameter—For DME, based on the DN, the query-condition parameter can be specified to fetch MOTL and ephemeral data with the following syntax: query-condition "rsp-foreign-subtree=applied-config"; query-condition "rsp-foreign-subtree=ephemeral".
		• For the YANG model, the sensor path format is as follows: *module_name*: *YANG_path*, where *module_name* is the name of the YANG model file. For example:
		• For device YANG:
		Cisco-NX-OS-device:System/bgp-items/inst-items
		• For OpenConfig YANG:
		openconfig-bgp:bgp
		Note The **depth**, **filter-condition**, and **query-condition** parameters are not supported for YANG currently.
		For the openconfig YANG models, go to YANG Models and navigate to the appropriate folder for the latest release.
		All the openconfig YANG models have a specific RPM, so you must install the associated RPM before you can use telemetry. See Adding Patch RPMs from Bash for more information on installing patch RPMs.

Step 12	**destination-group** *dgrp_id*
	Example:
	`switch(conf-tm-sensor)# destination-group 100`
	`switch(conf-tm-dest)#`
		Create a destination group and enter destination group configuration mode.
		Currently *dgrp_id* only supports numeric ID values.

	Command or Action	Purpose

Step 13	(Optional) **ip address** *ip_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	Example:
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50001 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50007 protocol HTTP encoding JSON`
	`switch(conf-tm-sensor)# ip address 171.70.55.69 port 50009 protocol UDP encoding JSON`
		Specify an IPv4 IP address and port to receive encoded telemetry data.
		Note gRPC is the default transport protocol.
		GPB is the default encoding.

Step 14	(Optional) **ipv6 address** *ipv6_address* **port** *port* **protocol** *procedural-protocol* **encoding** *encoding-protocol*
	Example:
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8000 protocol gRPC encoding GPB`
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8001 protocol HTTP encoding JSON`
	`switch(conf-tm-sensor)# ipv6 address 10:10::1 port 8002 protocol UDP encoding JSON`
		Specify an IPv6 IP address and port to receive encoded telemetry data.
		Note gRPC is the default transport protocol.
		GPB is the default encoding.

Step 15	*ip_version* **address** *ip_address* **port** *portnum*
	Example:
	• For IPv4:
	`switch(conf-tm-dest)# ip address 1.2.3.4 port 50003`
	• For IPv6:
	`switch(conf-tm-dest)# ipv6 address 10:10::1 port 8000`
		Create a destination profile for the outgoing data, where *ip_version* is either ip (for IPv4) or ipv6 (for IPv6).
		When the destination group is linked to a subscription, telemetry data is sent to the IP address and port that the profile specifies.

Step 16	(Optional) **use-chunking size** *chunking_size* **Example:** `switch(conf-tm-dest)# use-chunking size 64`	Enable gRPC chunking and set the chunking size, from 64 through 4096 bytes. See the section "Support for gRPC Chunking" for more information.

Step 17	**subscription** *sub_id*
	Example:
	`switch(conf-tm-dest)# subscription 100`
	`switch(conf-tm-sub)#`
		Create a subscription node with ID and enter the subscription configuration mode.
		Currently *sub_id* only supports numeric ID values.

	Command or Action	Purpose
		Note When subscribing to a DN, check whether the DN is supported by DME using REST to ensure that events can stream.
Step 18	**snsr-grp** *sgrp_id* **sample-interval** *interval* **Example:** `switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000`	Link the sensor group with ID *sgrp_id* to this subscription and set the data sampling interval in milliseconds. An interval value of 0 creates an event-based subscription, in which telemetry data is sent only upon changes under the specified MO. An interval value greater than 0 creates a frequency-based subscription, in which telemetry data is sent periodically at the specified interval. For example, an interval value of 15000 results in the sending of telemetry data every 15 seconds.
Step 19	**dst-grp** *dgrp_id* **Example:** `switch(conf-tm-sub)# dst-grp 100`	Link the destination group with ID *dgrp_id* to this subscription.

Configuration Examples for Telemetry Using the CLI

This example creates a subscription that streams data for the `sys/bgp` root MO every 5 seconds to the destination IP 1.2.3.4 port 50003.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a subscription that streams data for `sys/intf` every 5 seconds to destination IP 1.2.3.4 port 50003. The subscription encrypts the stream using GPB encoding that is verified using the `test.pem`.

```
switch(config)# telemetry
switch(config-telemetry)# certificate /bootflash/test.pem foo.test.google.fr
switch(conf-tm-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
switch(config-dest)# sensor-group 100
switch(conf-tm-sensor)# path sys/bgp depth 0
switch(conf-tm-sensor)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 5000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a subscription that streams data for `sys/cdp` every 15 seconds to destination IP 1.2.3.4 port 50004.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 15000
switch(conf-tm-sub)# dst-grp 100
```

This example creates a cadence-based collection of **show** command data every 750 seconds.

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 1
switch(conf-tm-dest)# ip address 172.27.247.72 port 60001 protocol gRPC encoding GPB
switch(conf-tm-dest)# sensor-group 1
switch(conf-tm-sensor# data-source NX-API
switch(conf-tm-sensor)# path "show system resources" depth 0
switch(conf-tm-sensor)# path "show version" depth 0
switch(conf-tm-sensor)# path "show environment power" depth 0
switch(conf-tm-sensor)# path "show environment fan" depth 0
switch(conf-tm-sensor)# path "show environment temperature" depth 0
switch(conf-tm-sensor)# path "show process cpu" depth 0
switch(conf-tm-sensor)# path "show nve peers" depth 0
switch(conf-tm-sensor)# path "show nve vni" depth 0
switch(conf-tm-sensor)# path "show nve vni 4002 counters" depth 0
switch(conf-tm-sensor)# path "show int nve 1 counters" depth 0
switch(conf-tm-sensor)# path "show policy-map vlan" depth 0
switch(conf-tm-sensor)# path "show ip access-list test" depth 0
switch(conf-tm-sensor)# path "show system internal access-list resource utilization" depth
 0
switch(conf-tm-sensor)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-dest)# snsr-grp 1 sample-interval 750000
```

This example creates an event-based subscription for `sys/fm`. Data is streamed to the destination only if there is a change under the sys/fm MO.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
```

During operation, you can change a sensor group from frequency-based to event-based, and change event-based to frequency-based by changing the sample-interval. This example changes the sensor-group from the previous example to frequency-based. After the following commands, the telemetry application will begin streaming the sys/fm data to the destination every 7 seconds.

```
switch(config)# telemetry
switch(config-telemetry)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
```

You can link multiple sensor groups and destinations to a single subscription. The subscription in this example streams the data for Ethernet port 1/1 to four different destinations every 10 seconds.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 1.2.3.4 port 50005
switch(conf-tm-sensor)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001 protocol HTTP encoding JSON
switch(conf-tm-dest)# ip address 1.4.8.2 port 60003
switch(conf-tm-dest)# subscription 100
switch(conf-tm-sub)# snsr-grp 100 sample-interval 10000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200
```

A sensor group can contain multiple paths. A destination group can contain multiple destination profiles. You can link a subscription to multiple sensor groups and destination groups, as shown in the following example.

```
switch(config)# telemetry
switch(config-telemetry)# sensor-group 100
switch(conf-tm-sensor)# path sys/intf/phys-[eth1/1] depth 0
switch(conf-tm-sensor)# path sys/epId-1 depth 0
switch(conf-tm-sensor)# path sys/bgp/inst/dom-default depth 0

switch(config-telemetry)# sensor-group 200
switch(conf-tm-sensor)# path sys/cdp depth 0
switch(conf-tm-sensor)# path sys/ipv4 depth 0

switch(config-telemetry)# sensor-group 300
switch(conf-tm-sensor)# path sys/fm depth 0
switch(conf-tm-sensor)# path sys/bgp depth 0

switch(conf-tm-sensor)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# ip address 4.3.2.5 port 50005

switch(conf-tm-dest)# destination-group 200
switch(conf-tm-dest)# ip address 5.6.7.8 port 50001

switch(conf-tm-dest)# destination-group 300
switch(conf-tm-dest)# ip address 1.2.3.4 port 60003

switch(conf-tm-dest)# subscription 600
switch(conf-tm-sub)# snsr-grp 100 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 200 sample-interval 20000
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 200

switch(conf-tm-dest)# subscription 900
switch(conf-tm-sub)# snsr-grp 200 sample-interval 7000
switch(conf-tm-sub)# snsr-grp 300 sample-interval 0
switch(conf-tm-sub)# dst-grp 100
switch(conf-tm-sub)# dst-grp 300
```

You can verify the telemetry configuration using the **show running-config telemetry** command, as shown in this example.

```
switch(config)# telemetry
switch(config-telemetry)# destination-group 100
switch(conf-tm-dest)# ip address 1.2.3.4 port 50003
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# end
switch# show run telemetry

!Command: show running-config telemetry
!Running configuration last done at: Wed Apr 12 21:29:11 2000
!Time: Wed Apr 12 21:29:23 2000

version 9.2(1) Bios:version 05.35
feature telemetry

telemetry

destination-group 100
ip address 1.2.3.4 port 50003 protocol gRPC encoding GPB
ip address 1.2.3.4 port 50004 protocol gRPC encoding GPB
```

You can specify transport VRF and telemetry data compression for gRPC using the **use-vrf** and **use-compression gzip** commands, as shown in this example.

```
switch(config)# telemetry
switch(config-telemetry)# destination-profile
switch(conf-tm-dest-profile)# use-vrf default
switch(conf-tm-dest-profile)# use-compression gzip
switch(conf-tm-dest-profile)# sensor-group 1
switch(conf-tm-sensor)# path sys/bgp depth unbounded
switch(conf-tm-sensor)# destination-group 1
switch(conf-tm-dest)# ip address 1.2.3.4 port 50004
switch(conf-tm-dest)# subscription 1
switch(conf-tm-sub)# dst-grp 1
switch(conf-tm-sub)# snsr-grp 1 sample-interval 10000
```

Displaying Telemetry Configuration and Statistics

Use the following NX-OS CLI **show** commands to display telemetry configuration, statistics, errors, and session information.

show telemetry control database

This command displays the internal databases that reflect the configuration of telemetry.

```
switch# show telemetry control database ?
  <CR>
  >                   Redirect it to a file
  >>                  Redirect it to a file in append mode
  destination-groups  Show destination-groups
  destinations        Show destinations
  sensor-groups       Show sensor-groups
  sensor-paths        Show sensor-paths
  subscriptions       Show subscriptions
  |                   Pipe command output to filter

switch# show telemetry control database

Subscription Database size = 1
```

```
--------------------------------------------------------------------------------
Subscription ID      Data Collector Type
--------------------------------------------------------------------------------
100                  DME NX-API

Sensor Group Database size = 1

--------------------------------------------------------------------------------
Sensor Group ID  Sensor Group type  Sampling interval(ms)  Linked subscriptions
--------------------------------------------------------------------------------
100              Timer              10000(Running)         1

Sensor Path Database size = 1

--------------------------------------------------------------------------------
Subscribed Query Filter  Linked Groups  Sec Groups  Retrieve level  Sensor Path
--------------------------------------------------------------------------------
No                       1              0           Full            sys/fm

Destination group Database size = 2

--------------------------------------------------------------------------------
Destination Group ID  Refcount
--------------------------------------------------------------------------------
100                    1

Destination Database size = 2

--------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
--------------------------------------------------------------------------------
192.168.20.111       12345      JSON       HTTP       1
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry control stats

This command displays the statistics about the internal databases about configuration of telemetry.

```
switch# show telemetry control stats
show telemetry control stats entered

--------------------------------------------------------------------------------
Error Description                                              Error Count
--------------------------------------------------------------------------------
Chunk allocation failures                                      0
Sensor path Database chunk creation failures                   0
Sensor Group Database chunk creation failures                  0
Destination Database chunk creation failures                   0
Destination Group Database chunk creation failures             0
Subscription Database chunk creation failures                  0
Sensor path Database creation failures                         0
Sensor Group Database creation failures                        0
Destination Database creation failures                         0
Destination Group Database creation failures                   0
Subscription Database creation failures                        0
Sensor path Database insert failures                           0
Sensor Group Database insert failures                          0
Destination Database insert failures                           0
Destination Group Database insert failures                     0
Subscription insert to Subscription Database failures          0
Sensor path Database delete failures                           0
```

```
Sensor Group Database delete failures                        0
Destination Database delete failures                         0
Destination Group Database delete failures                   0
Delete Subscription from Subscription Database failures      0
Sensor path delete in use                                    0
Sensor Group delete in use                                   0
Destination delete in use                                    0
Destination Group delete in use                              0
Delete destination(in use) failure count                     0
Failed to get encode callback                                0
Sensor path Sensor Group list creation failures              0
Sensor path prop list creation failures                      0
Sensor path sec Sensor path list creation failures           0
Sensor path sec Sensor Group list creation failures          0
Sensor Group Sensor path list creation failures              0
Sensor Group Sensor subs list creation failures              0
Destination Group subs list creation failures                0
Destination Group Destinations list creation failures        0
Destination Destination Groups list creation failures        0
Subscription Sensor Group list creation failures             0
Subscription Destination Groups list creation failures       0
Sensor Group Sensor path list delete failures                0
Sensor Group Subscriptions list delete failures              0
Destination Group Subscriptions list delete failures         0
Destination Group Destinations list delete failures          0
Subscription Sensor Groups list delete failures              0
Subscription Destination Groups list delete failures         0
Destination Destination Groups list delete failures          0
Failed to delete Destination from Destination Group          0
Failed to delete Destination Group from Subscription         0
Failed to delete Sensor Group from Subscription              0
Failed to delete Sensor path from Sensor Group               0
Failed to get encode callback                                0
Failed to get transport callback                             0
switch#  Destination Database size = 1

-------------------------------------------------------------------------------
Dst IP Addr     Dst Port   Encoding   Transport  Count
-------------------------------------------------------------------------------
192.168.20.123 50001      GPB        gRPC       1
```

show telemetry data collector brief

This command displays the brief statistics about the data collection.

```
switch# show telemetry data collector brief

-----------------------------------------------------------------------
Collector Type       Successful Collections     Failed Collections
-----------------------------------------------------------------------
DME                   143                        0
```

show telemetry data collector details

This command displays detailed statistics about the data collection which includes breakdown of all sensor paths.

```
switch# show telemetry data collector details
```

```
-------------------------------------------------------------------------------
Succ Collections      Failed Collections     Sensor Path
-------------------------------------------------------------------------------
150                   0                      sys/fm
```

show telemetry event collector errors

This command displays the errors statistic about the event collection.

```
switch# show telemetry event collector errors

-------------------------------------------------------------------------------
Error Description                                Error Count
-------------------------------------------------------------------------------
APIC-Cookie Generation Failures                  - 0
Authentication Failures                          - 0
Authentication Refresh Failures                  - 0
Authentication Refresh Timer Start Failures      - 0
Connection Timer Start Failures                  - 0
Connection Attempts                              - 3
Dme Event Subscription Init Failures             - 0
Event Data Enqueue Failures                      - 0
Event Subscription Failures                      - 0
Event Subscription Refresh Failures              - 0
Pending Subscription List Create Failures        - 0
Subscription Hash Table Create Failures          - 0
Subscription Hash Table Destroy Failures         - 0
Subscription Hash Table Insert Failures          - 0
Subscription Hash Table Remove Failures          - 0
Subscription Refresh Timer Start Failures        - 0
Websocket Connect Failures                       - 0
```

show telemetry event collector stats

This command displays the statistics about the event collection which includes breakdown of all sensor paths.

```
switch# show telemetry event collector stats

-------------------------------------------------------------------------------
Collection Count  Latest Collection Time    Sensor Path
-------------------------------------------------------------------------------
```

show telemetry control pipeline stats

This command displays the statistics for the telemetry pipeline.

```
switch# show telemetry pipeline stats
Main Statistics:
    Timers:
        Errors:
            Start Fail        =     0

    Data Collector:
        Errors:
            Node Create Fail  =     0

    Event Collector:
```

```
        Errors:
            Node Create Fail  =     0    Node Add Fail     =     0
            Invalid Data      =     0

Queue Statistics:
    Request Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

    Data Queue:
        High Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0

        Low Priority Queue:
            Info:
                Actual Size       =    50    Current Size      =     0
                Max Size          =     0    Full Count        =     0

            Errors:
                Enqueue Error     =     0    Dequeue Error     =     0
```

show telemetry transport

This command displays all configured transport sessions.

```
switch# show telemetry transport

Session Id      IP Address      Port       Encoding   Transport  Status
--------------------------------------------------------------------------------
0               192.168.20.123   50001      GPB        gRPC       Connected

Retry buffer Size:             10485760
Event Retry Messages (Bytes):  0
Timer Retry Messages (Bytes):  0
Total Retries sent:            0
Total Retries Dropped:         0
```

show telemetry transport <session-id>

This command displays detailed session information for a specific transport session.

```
switch# show telemetry transport 0
```

```
Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           gRPC
Status:              Disconnected
Last Connected:      Fri Sep 02 11:45:57.505 UTC
Tx Error Count:      224
Last Tx Error:       Fri Sep 02 12:23:49.555 UTC

switch# show telemetry transport 1

Session Id:          1
IP Address:Port      10.30.218.56:51235
Encoding:            JSON
Transport:           HTTP
Status:              Disconnected
Last Connected:      Never
Last Disconnected:   Never
Tx Error Count:      3
Last Tx Error:       Wed Apr 19 15:56:51.617 PDT
```

The following example shows output from an IPv6 entry.

```
switch# show telemetry transport 0
Session Id: 0
IP Address:Port [10:10::1]:8000
Transport: GRPC
Status: Idle
Last Connected: Never
Last Disconnected: Never
Tx Error Count: 0
Last Tx Error: None
Event Retry Queue Bytes: 0
Event Retry Queue Size: 0
Timer Retry Queue Bytes: 0
Timer Retry Queue Size: 0
Sent Retry Messages: 0
Dropped Retry Messages: 0
```

show telemetry transport <session-id> stats

This command displays details of a specific transport session.

```
switch# show telemetry transport 0 stats

Session Id:          0
IP Address:Port      192.168.20.123:50001
Encoding:            GPB
Transport:           GRPC
Status:              Connected
Last Connected:      Mon May 01 11:29:46.912 PST
Last Disconnected:   Never
Tx Error Count:      0
Last Tx Error:       None
```

show telemetry transport <session-id> errors

This command displays detailed error statistics for a specific transport session.

```
switch# show telemetry transport 0 errors

Session Id:                   0
Connection Stats
   Connection Count           1
   Last Connected:            Mon May 01 11:29:46.912 PST
   Disconnect Count           0
   Last Disconnected:         Never
Transmission Stats
   Transmit Count:            1225
   Last TX time:              Tue May 02 11:40:03.531 PST
   Min Tx Time:               7                  ms
   Max Tx Time:               1760               ms
   Avg Tx Time:               500                ms
```

Displaying Telemetry Log and Trace Information

Use the following NX-OS CLI commands to display the log and trace information.

show tech-support telemetry

This NX-OS CLI command collects the telemetry log contents from the tech-support log. In this example, the command output is redirected into a file in bootflash.

```
switch# show tech-support telemetry > bootflash:tmst.log
```

show system internal telemetry trace

The **show system internal telemetry trace** [**tm-events** | **tm-errors** |**tm-logs** | **all**] command displays system internal telemetry trace information.

```
switch# show system internal telemetry trace all
Telemetry All Traces:
Telemetry Error Traces:
[07/26/17 15:22:29.156 UTC 1 28577] [3960399872][tm_cfg_api.c:367] Not able to destroy dest
 profile list for config node rc:-1610612714 reason:Invalid argument
[07/26/17 15:22:44.972 UTC 2 28577] [3960399872][tm_stream.c:248] No subscriptions for
destination group 1
[07/26/17 15:22:49.463 UTC 3 28577] [3960399872][tm_stream.c:576] TM_STREAM: Subscriptoin
1 does not have any sensor groups

3 entries printed
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
 initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
switch#

switch# show system internal telemetry trace tm-logs
Telemetry Log Traces:
0 entries printed
switch#
switch# show system internal telemetry trace tm-events
```

```
Telemetry Event Traces:
[07/26/17 15:19:40.610 UTC 1 28577] [3960399872][tm_debug.c:41] Telemetry xostrace buffers
 initialized successfully!
[07/26/17 15:19:40.610 UTC 2 28577] [3960399872][tm.c:744] Telemetry statistics created
successfully!
[07/26/17 15:19:40.610 UTC 3 28577] [3960399872][tm_init_n9k.c:97] Platform intf:
grpc_traces:compression,channel
[07/26/17 15:19:40.610 UTC 4 28577] [3960399872][tm_init_n9k.c:207] Adding telemetry to
cgroup
[07/26/17 15:19:40.670 UTC 5 28577] [3960399872][tm_init_n9k.c:215] Added telemetry to
cgroup successfully!

switch# show system internal telemetry trace tm-errors
Telemetry Error Traces:
0 entries printed
switch#
```

Configuring Telemetry Using the NX-API

Configuring Telemetry Using the NX-API

In the object model of the switch DME, the configuration of the telemetry feature is defined in a hierarchical structure of objects as shown in Telemetry Model in the DME, on page 36. Following are the main objects to be configured:

- **fmEntity** — Contains the NX-API and Telemetry feature states.
  - **fmNxapi** — Contains the NX-API state.
  - **fmTelemetry** — Contains the Telemetry feature state.
- **telemetryEntity** — Contains the telemetry feature configuration.
  - **telemetrySensorGroup** — Contains the definitions of one or more sensor paths or nodes to be monitored for telemetry. The telemetry entity can contain one or more sensor groups.
    - **telemetryRtSensorGroupRel** — Associates the sensor group with a telemetry subscription.
    - **telemetrySensorPath** — A path to be monitored. The sensor group can contain multiple objects of this type.
  - **telemetryDestGroup** — Contains the definitions of one or more destinations to receive telemetry data. The telemetry entity can contain one or more destination groups.
    - **telemetryRtDestGroupRel** — Associates the destination group with a telemetry subscription.
    - **telemetryDest** — A destination address. The destination group can contain multiple objects of this type.
  - **telemetrySubscription** — Specifies how and when the telemetry data from one or more sensor groups is sent to one or more destination groups.
    - **telemetryRsDestGroupRel** — Associates the telemetry subscription with a destination group.
    - **telemetryRsSensorGroupRel** — Associates the telemetry subscription with a sensor group.

- **telemetryCertificate** — Associates the telemetry subscription with a certificate and hostname.

To configure telemetry using the NX-API, you construct a JSON representation of the telemetry object structure and push it to the DME with an HTTP or HTTPS POST operation.

Note For detailed instructions on using the NX-API, see the *Cisco Nexus 3000 and 9000 Series NX-API REST SDK User Guide and API Reference*.

Before you begin

Your switch must be configured to run the NX-API from the CLI:

```
switch(config)# feature nxapi

nxapi use-vrf vrf_name
nxapi http port port_number
```

Procedure

	Command or Action	Purpose
Step 1	Enable the telemetry feature. **Example:** `{ "fmEntity" : { "children" : [{ "fmTelemetry" : { "attributes" : { "adminSt" : "enabled" } } } ] } }`	The root element is **fmTelemetry** and the base path for this element is `sys/fm`. Configure the **adminSt** attribute as `enabled`.
Step 2	Create the root level of the JSON payload to describe the telemetry configuration. **Example:** `{ "telemetryEntity": { "attributes": { "dn": "sys/tm" }, } }`	The root element is **telemetryEntity** and the base path for this element is `sys/tm`. Configure the **dn** attribute as `sys/tm`.

	Command or Action	Purpose
Step 3	Create a sensor group to contain the defined sensor paths. **Example:** `"telemetrySensorGroup": { "attributes": { "id": "10", "rn": "sensor-10" "dataSrc": "NX-API" }, "children": [{ }] }`	A telemetry sensor group is defined in an object of class **telemetrySensorGroup**. Configure the following attributes of the object: • **id** — An identifier for the sensor group. Currently only numeric ID values are supported. • **rn** — The relative name of the sensor group object in the format: **sensor-***id*. • **dataSrc** — Selects the data source from **DEFAULT**, **DME**, **YANG**, or **NX-API**. Children of the sensor group object will include sensor paths and one or more relation objects (**telemetryRtSensorGroupRel**) to associate the sensor group with a telemetry subscription.
Step 4	(Optional) Add an SSL/TLS certificate and a host. **Example:** `{ "telemetryCertificate": { "attributes": { "filename": "root.pem" "hostname": "c.com" } } }`	The **telemetryCertificate** defines the location of the SSL/TLS certificate with the telemetry subscription/destination.
Step 5	Define a telemetry destination group. **Example:** `{ "telemetryDestGroup": { "attributes": { "id": "20" } } }`	A telemetry destination group is defined in **telemetryEntity**. Configure the id attribute.
Step 6	Define a telemetry destination profile. **Example:** `{ "telemetryDestProfile": { "attributes": { "adminSt": "enabled" }, "children": [ {`	A telemetry destination profile is defined in **telemetryDestProfile**. • Configure the **adminSt** attribute as `enabled`. • Under **telemetryDestOptSourceInterface**, configure the **name** attribute with an interface name to stream data from the

	Command or Action	Purpose

	<pre>"telemetryDestOptSourceInterface": {
	"attributes": {
	"name": "lo0"
	}
	}
	}
	]
	}
	}</pre>
		configured interface to a destination with the source IP address.

Step 7	Define one or more telemetry destinations, consisting of an IP address and port number to which telemetry data will be sent.
	Example:
	<pre>{
	"telemetryDest": {
	"attributes": {
	"addr": "1.2.3.4",
	"enc": "GPB",
	"port": "50001",
	"proto": "gRPC",
	"rn":
	"addr-[1.2.3.4]-port-50001"
	}
	}
	}</pre>
		A telemetry destination is defined in an object of class **telemetryDest**. Configure the following attributes of the object:
		• **addr** — The IP address of the destination.
		• **port** — The port number of the destination.
		• **rn** — The relative name of the destination object in the format: **path-[***path***]**.
		• **enc** — The encoding type of the telemetry data to be sent. NX-OS supports:
		  • Google protocol buffers (GPB) for gRPC.
		  • JSON for C.
		  • GPB or JSON for UDP and secure UDP (DTLS).
		• **proto** — The transport protocol type of the telemetry data to be sent. NX-OS supports:
		  • gRPC
		  • HTTP
		  • VUDP and secure UDP (DTLS)

Step 8	Enable gRPC chunking and set the chunking size, between 64 and 4096 bytes.
	Example:
	<pre>{
	"telemetryDestGrpOptChunking": {
	"attributes": {
	"chunkSize": "2048",
	"dn":
	"sys/tm/dest-1/chunking"
	}</pre>
		See "Support for gRPC Chunking" in Guidelines and Limitations, on page 4 for more information.

	Command or Action	Purpose
	`    }` `}`	

Step 9	Create a telemetry subscription to configure the telemetry behavior.
	Example:
	`"telemetrySubscription": {`
	`  "attributes": {`
	`  "id": "30",`
	`  "rn": "subs-30"`
	`  }, "children": [{`
	`  }]`
	`}`
		A telemetry subscription is defined in an object of class **telemetrySubscription**. Configure the following attributes of the object:
		• **id** — An identifier for the subscription. Currently only numeric ID values are supported.
		• **rn** — The relative name of the subscription object in the format: **subs-***id*.
		Children of the subscription object will include relation objects for sensor groups (**telemetryRsSensorGroupRel**) and destination groups (**telemetryRsDestGroupRel**).

Step 10	Add the sensor group object as a child object to the **telemetrySubscription** element under the root element (**telemetryEntity**).
	Example:
	`{`
	`  "telemetrySubscription": {`
	`  "attributes": {`
	`  "id": "30"`
	`  }`
	`  "children": [{`
	`  "telemetryRsSensorGroupRel":`
	`{`
	`  "attributes": {`
	`  "sampleIntvl": "5000",`
	`  "tDn": "sys/tm/sensor-10"`
	`  }`
	`  }`
	`  }`
	`  ]`
	`  }`
	`  }`	

Step 11	Create a relation object as a child object of the subscription to associate the subscription to the telemetry sensor group and to specify the data sampling behavior.
	Example:
	`"telemetryRsSensorGroupRel": {`
	`  "attributes": {`
	`  "rType": "mo",`
	`  "rn":`
	`"rssensorGroupRel-[sys/tm/sensor-10]",`
		The relation object is of class **telemetryRsSensorGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rssensorGroupRel-[sys/tm/***sensor-group-id***]**.
		• **sampleIntvl** — The data sampling period in milliseconds. An interval value of 0

	Command or Action	Purpose

	<pre>"sampleIntvl": "5000",
	"tCl": "telemetrySensorGroup",
	"tDn": "sys/tm/sensor-10",
	"tType": "mo"
	}
	}</pre>
		creates an event-based subscription, in which telemetry data is sent only upon changes under the specified MO. An interval value greater than 0 creates a frequency-based subscription, in which telemetry data is sent periodically at the specified interval. For example, an interval value of 15000 results in the sending of telemetry data every 15 seconds.
		• **tCl** — The class of the target (sensor group) object, which is **telemetrySensorGroup**.
		• **tDn** — The distinguished name of the target (sensor group) object, which is **sys/tm/***sensor-group-id*.
		• **rType** — The relation type, which is **mo** for managed object.
		• **tType** — The target type, which is **mo** for managed object.

Step 12	Define one or more sensor paths or nodes to be monitored for telemetry.
	Example:
	Single sensor path
	<pre>{
	"telemetrySensorPath": {
	"attributes": {
	"path": "sys/cdp",
	"rn": "path-[sys/cdp]",
	"excludeFilter": "",
	"filterCondition": "",
	"path": "sys/fm/bgp",
	"secondaryGroup": "0",
	"secondaryPath": "",
	"depth": "0"
	}
	}
	}</pre>**Example:**
	Single sensor path for NX-API
	<pre>{
	"telemetrySensorPath": {
	"attributes": {
	"path": "show interface",
	"path": "show bgp",</pre>
		A sensor path is defined in an object of class **telemetrySensorPath**. Configure the following attributes of the object:
		• **path** — The path to be monitored.
		• **rn** — The relative name of the path object in the format: **path-[***path***]**
		• **depth** — The retrieval level for the sensor path. A depth setting of **0** retrieves only the root MO properties.
		• **filterCondition** — (Optional) Creates a specific filter for event-based subscriptions. The DME provides the filter expressions. For more information about filtering, see the Cisco APIC REST API Usage Guidelines on composing queries: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/2-x/rest_cfg/2_1_x/b_Cisco_APIC_REST_API_Configuration_Guide/b_Cisco_APIC_REST_API_Configuration_Guide_chapter_01.html#d25e1534a1635

	Command or Action	Purpose

```
            "rn": "path-[sys/cdp]",
            "excludeFilter": "",
            "filterCondition": "",
            "path": "sys/fm/bgp",
            "secondaryGroup": "0",
            "secondaryPath": "",
            "depth": "0"
        }
    }
}
```

Example:

Multiple sensor paths

```
{
    "telemetrySensorPath": {
        "attributes": {
            "path": "sys/cdp",
            "rn": "path-[sys/cdp]",
            "excludeFilter": "",
            "filterCondition": "",
            "path": "sys/fm/bgp",
            "secondaryGroup": "0",
            "secondaryPath": "",
            "depth": "0"
        }
    }
},
{
     "telemetrySensorPath": {
         "attributes": {
            "excludeFilter": "",
            "filterCondition": "",
            "path": "sys/fm/dhcp",
            "secondaryGroup": "0",
            "secondaryPath": "",
            "depth": "0"
        }
    }
}
```

Example:

Single sensor path filtering for BGP disable events:

```
{
    "telemetrySensorPath": {
        "attributes": {
            "path": "sys/cdp",
            "rn": "path-[sys/cdp]",
            "excludeFilter": "",
            "filterCondition":
"eq(fmBgp.operSt.\"disabled\")",
            "path": "sys/fm/bgp",
            "secondaryGroup": "0",
            "secondaryPath": "",
            "depth": "0"
        }
```

	Command or Action	Purpose
	<pre> } }</pre>	
Step 13	Add sensor paths as child objects to the sensor group object (**telemetrySensorGroup**).	
Step 14	Add destinations as child objects to the destination group object (**telemetryDestGroup**).	
Step 15	Add the destination group object as a child object to the root element (**telemetryEntity**).	

Step 16	Create a relation object as a child object of the telemetry sensor group to associate the sensor group to the subscription.
	Example:
	<pre>"telemetryRtSensorGroupRel": {
	    "attributes": {
	        "rn":
	"rtsensorGroupRel-[sys/tm/subs-30]",
	        "tCl": "telemetrySubscription",
	
	        "tDn": "sys/tm/subs-30"
	    }
	}</pre>
		The relation object is of class **telemetryRtSensorGroupRel** and is a child object of **telemetrySensorGroup**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rtsensorGroupRel-[sys/tm/***subscription-id***]**.
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

Step 17	Create a relation object as a child object of the telemetry destination group to associate the destination group to the subscription.
	Example:
	<pre>"telemetryRtDestGroupRel": {
	    "attributes": {
	        "rn":
	"rtdestGroupRel-[sys/tm/subs-30]",
	        "tCl": "telemetrySubscription",
	
	        "tDn": "sys/tm/subs-30"
	    }
	}</pre>
		The relation object is of class **telemetryRtDestGroupRel** and is a child object of **telemetryDestGroup**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rtdestGroupRel-[sys/tm/***subscription-id***]**.
		• **tCl** — The target class of the subscription object, which is **telemetrySubscription**.
		• **tDn** — The target distinguished name of the subscription object, which is **sys/tm/***subscription-id*.

Step 18	Create a relation object as a child object of the subscription to associate the subscription to the telemetry destination group.
	Example:
	<pre>"telemetryRsDestGroupRel": {
	    "attributes": {
	        "rType": "mo",</pre>
		The relation object is of class **telemetryRsDestGroupRel** and is a child object of **telemetrySubscription**. Configure the following attributes of the relation object:
		• **rn** — The relative name of the relation object in the format: **rsdestGroupRel-[sys/tm/***destination-group-id***]**.

	Command or Action	Purpose
	<pre>"rn": "rsdestGroupRel-[sys/tm/dest-20]", "tCl": "telemetryDestGroup", "tDn": "sys/tm/dest-20", "tType": "mo" } }</pre>	• **tCl** — The class of the target (destination group) object, which is **telemetryDestGroup**. • **tDn** — The distinguished name of the target (destination group) object, which is **sys/tm/***destination-group-id*. • **rType** — The relation type, which is **mo** for managed object. • **tType** — The target type, which is **mo** for managed object.
Step 19	Send the resulting JSON structure as an HTTP/HTTPS POST payload to the NX-API endpoint for telemetry configuration.	The base path for the telemetry entity is `sys/tm` and the NX-API endpoint is: `{{URL}}/api/node/mo/sys/tm.json`

Example

The following is an example of all the previous steps collected into one POST payload (note that some attributes may not match):

```
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
              "addr": "10.30.217.80",
              "port": "50051",
              "enc": "GPB",
```

```
            "proto": "gRPC"
          }
        }
      }
      ]
    }
  },
  {
    "telemetrySubscription": {
      "attributes": {
        "id": "30"
      }
      "children": [{
        "telemetryRsSensorGroupRel": {
          "attributes": {
            "sampleIntvl": "5000",
            "tDn": "sys/tm/sensor-10"
          }
        }
      },
      {
        "telemetryRsDestGroupRel": {
          "attributes": {
            "tDn": "sys/tm/dest-20"
          }
        }
      }
      ]
    }
  }
  ]
 }
}
```

Configuration Example for Telemetry Using the NX-API

Streaming Paths to a Destination

This example creates a subscription that streams paths `sys/cdp` and `sys/ipv4` to a destination `1.2.3.4 port 50001` every five seconds.

```
POST https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
    "telemetryEntity": {
        "attributes": {
            "dn": "sys/tm"
        },
        "children": [{
            "telemetrySensorGroup": {
                "attributes": {
                    "id": "10",
                    "rn": "sensor-10"
                },  "children": [{
                    "telemetryRtSensorGroupRel": {
                        "attributes": {
                            "rn": "rtsensorGroupRel-[sys/tm/subs-30]",
                            "tCl": "telemetrySubscription",
                            "tDn": "sys/tm/subs-30"
```

```
                    }
                }
            }, {
                "telemetrySensorPath": {
                    "attributes": {
                        "path": "sys/cdp",
                        "rn": "path-[sys/cdp]",
                        "excludeFilter": "",
                        "filterCondition": "",
                        "secondaryGroup": "0",
                        "secondaryPath": "",
                        "depth": "0"
                    }
                }
            }, {
                "telemetrySensorPath": {
                    "attributes": {
                        "path": "sys/ipv4",
                        "rn": "path-[sys/ipv4]",
                        "excludeFilter": "",
                        "filterCondition": "",
                        "secondaryGroup": "0",
                        "secondaryPath": "",
                        "depth": "0"
                    }
                }
            }]
        }
    }, {
        "telemetryDestGroup": {
            "attributes": {
                "id": "20",
                "rn": "dest-20"
            },
            "children": [{
                "telemetryRtDestGroupRel": {
                    "attributes": {
                        "rn": "rtdestGroupRel-[sys/tm/subs-30]",
                        "tCl": "telemetrySubscription",
                        "tDn": "sys/tm/subs-30"
                    }
                }
            }, {
                "telemetryDest": {
                    "attributes": {
                        "addr": "1.2.3.4",
                        "enc": "GPB",
                        "port": "50001",
                        "proto": "gRPC",
                        "rn": "addr-[1.2.3.4]-port-50001"
                    }
                }
            }]
        }
    }, {
        "telemetrySubscription": {
            "attributes": {
                "id": "30",
                "rn": "subs-30"
            },
            "children": [{
                "telemetryRsDestGroupRel": {
                    "attributes": {
                        "rType": "mo",
```

```
                            "rn": "rsdestGroupRel-[sys/tm/dest-20]",
                            "tCl": "telemetryDestGroup",
                            "tDn": "sys/tm/dest-20",
                            "tType": "mo"
                        }
                    }
                }, {
                    "telemetryRsSensorGroupRel": {
                        "attributes": {
                            "rType": "mo",
                            "rn": "rssensorGroupRel-[sys/tm/sensor-10]",
                            "sampleIntvl": "5000",
                            "tCl": "telemetrySensorGroup",
                            "tDn": "sys/tm/sensor-10",
                            "tType": "mo"
                        }
                    }
                }]
            }
        }]
    }
}
```

Filter Conditions on BGP Notifications

The following example payload enables notifications that trigger when the BFP feature is disabled as per the `filterCondition` attribute in the `telemetrySensorPath` MO. The data is streamed to`10.30.217.80 port 50055`.

```
POST  https://192.168.20.123/api/node/mo/sys/tm.json

Payload:
{
  "telemetryEntity": {
    "children": [{
      "telemetrySensorGroup": {
        "attributes": {
          "id": "10"
        }
        "children": [{
          "telemetrySensorPath": {
            "attributes": {
              "excludeFilter": "",
              "filterCondition": "eq(fmBgp.operSt,\"disabled\")",
              "path": "sys/fm/bgp",
              "secondaryGroup": "0",
              "secondaryPath": "",
              "depth": "0"
            }
          }
        }
        ]
      }
    },
    {
      "telemetryDestGroup": {
        "attributes": {
          "id": "20"
        }
        "children": [{
          "telemetryDest": {
            "attributes": {
```

```
              "addr": "10.30.217.80",
              "port": "50055",
              "enc": "GPB",
              "proto": "gRPC"
            }
          }
        }
        ]
      }
    },
    {
      "telemetrySubscription": {
        "attributes": {
          "id": "30"
        }
        "children": [{
          "telemetryRsSensorGroupRel": {
            "attributes": {
              "sampleIntvl": "0",
              "tDn": "sys/tm/sensor-10"
            }
          }
        },
        {
          "telemetryRsDestGroupRel": {
            "attributes": {
              "tDn": "sys/tm/dest-20"
            }
          }
        }
        ]
      }
    }
    ]
  }
}
```

Using Postman Collection for Telemetry Configuration

An example Postman collection is an easy way to start configuring the telemetry feature, and can run all telemetry CLI equivalents in a single payload. Modify the file in the preceding link using your preferred text editor to update the payload to your needs, then open the collection in Postman and run the collection.

Telemetry Model in the DME

The telemetry application is modeled in the DME with the following structure:

```
model
|----package [name:telemetry]
     |  @name:telemetry
     |----objects
          |----mo [name:Entity]
               |    @name:Entity
               |      @label:Telemetry System
               |--property
               |    @name:adminSt
               |      @type:AdminState
               |
               |----mo [name:SensorGroup]
               |    |    @name:SensorGroup
               |    |      @label:Sensor Group
```

```
|    |--property
|    |    @name:id [key]
|    |      @type:string:Basic
|    |    @name:dataSrc
|    |      @type:DataSource
|    |
|    |----mo [name:SensorPath]
|         |   @name:SensorPath
|         |     @label:Sensor Path
|         |--property
|         |    @name:path [key]
|         |      @type:string:Basic
|         |    @name:filterCondition
|         |      @type:string:Basic
|         |    @name:excludeFilter
|         |      @type:string:Basic
|         |    @name:depth
|         |      @type:RetrieveDepth
|
|----mo [name:DestGroup]
|    |    @name:DestGroup
|    |      @label:Destination Group
|    |--property
|    |    @name:id
|    |      @type:string:Basic
|    |
|    |----mo [name:Dest]
|         |   @name:Dest
|         |     @label:Destination
|         |--property
|         |    @name:addr [key]
|         |      @type:address:Ip
|         |    @name:port [key]
|         |      @type:scalar:Uint16
|         |    @name:proto
|         |      @type:Protocol
|         |    @name:enc
|         |      @type:Encoding
|
|----mo [name:Subscription]
     |    @name:Subscription
     |      @label:Subscription
     |--property
     |    @name:id
     |      @type:scalar:Uint64
     |----reldef
     |    |  @name:SensorGroupRel
     |    |    @to:SensorGroup
     |    |    @cardinality:ntom
     |    |    @label:Link to sensorGroup entry
     |    |--property
     |         @name:sampleIntvl
     |           @type:scalar:Uint64
     |
     |----reldef
          |  @name:DestGroupRel
          |    @to:DestGroup
          |    @cardinality:ntom
          |    @label:Link to destGroup entry
```