Configure Model-driven Telemetry

Model-driven Telemetry (MDT) provides a mechanism to stream data from an MDT-capable device to a destination. The data to be streamed is defined through subscription.

The data to be streamed is subscribed from a data set in a YANG model. The data from the subscribed data set is streamed out to the destination either at a configured periodic interval or only when an event occurs. This behavior is based on whether MDT is configured for cadence-based telemetry or event-based telemetry (EDT).

The configuration for event-based telemetry is similar to cadence-based telemetry, with only the sample interval as the differentor. Configuring the sample interval value to zero sets the subscription for event-based telemetry, while configuring the interval to any non-zero value sets the subscription for cadence-based telemetry.

The following YANG models are used to configure and monitor MDT:

- **Cisco-IOS-XR-telemetry-model-driven-cfg.yang** and **openconfig-telemetry.yang:** configure MDT using NETCONF or merge-config over grpc.
- **Cisco-IOS-XR-telemetry-model-driven-oper.yang:** get the operational information about MDT.

For the nodes that support event-driven telemetry (EDT), the YANG model is annotated with the statement `xr:event-telemetry`. For example, the interface that supports EDT has an annotation as shown in the following example:

```
leaf interface-name {
      xr:event-telemetry "Subscribe Telemetry Event";
      type xr:Interface-name;
      description "Member's interface name";
    }
```

The process of streaming MDT data uses these components:

- **Destination:** specifies one or more destinations to collect the streamed data.
- **Sensor path:** specifies the YANG path from which data has to be streamed.
- **Subscription:** binds one or more sensor-paths to destinations, and specifies the criteria to stream data. In cadence-based telemetry, data is streamed continuously at a configured frequency. In event-based telemetry, data is streamed only when a change in the state or data for the configured model occurs.
- **Transport and encoding:** represents the delivery mechanism of telemetry data.

The options to initialize a telemetry session between the router and destination is based on two modes:

- Dial-out mode: The router initiates a session to the destinations based on the subscription.
- Dial-in mode: The destination initiates a session to the router and subscribes to data to be streamed.

Note Dial-in mode is supported only over gRPC.

Streaming model-driven telemetry data to the intended receiver involves these tasks:

- Configure Dial-out Mode, on page 2
- Configure Dial-in Mode, on page 9
- Event-driven Telemetry for Terminal-device Models, on page 13
- Streaming Event-Driven Telemetry for Online Insertion and Removal of Pluggables, on page 14
- gRPC Network Management Interface, on page 25
- 2s Telemetry Based on GNMI Subscribe, on page 26
- gNMI Heartbeat Interval, on page 27

Configure Dial-out Mode

In a dial-out mode, the router initiates a session to the destinations based on the subscription.

All 64-bit IOS XR platforms (except for NCS 6000 series routers) support gRPC , UDP and TCP protocols. All 32-bit IOS XR platforms support only TCP.

For more information about the dial-out mode, see Dial-out Mode.

The process to configure a dial-out mode involves:

Create a Destination Group

The destination group specifies the destination address, port, encoding and transport that the router uses to send out telemetry data.

1. Identify the destination address, port, transport, and encoding format.
2. Create a destination group.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group <group-name>

Router(config-model-driven-dest)#address family ipv4 <IP-address> port <port-number>
Router(config-model-driven-dest-addr)#encoding <encoding-format>
Router(config-model-driven-dest-addr)#protocol <transport>
Router(config-model-driven-dest-addr)#commit
```

Example: Destination Group for TCP Dial-out

The following example shows a destination group `DGroup1` created for TCP dial-out configuration with key-value Google Protocol Buffers (also called self-describing-gpb) encoding:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group DGroup1
```

```
Router(config-model-driven-dest)#address family ipv4 172.0.0.0 port 5432
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
```

Example: Destination Group for UDP Dial-out

The following example shows a destination group `DGroup1` created for UDP dial-out configuration with key-value Google Protocol Buffers (also called self-describing-gpb) encoding:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group DGroup1
Router(config-model-driven-dest)#address family ipv4 172.0.0.0 port 5432
Router(config-model-driven-dest-addr)#encoding self-describing-gpb
Router(config-model-driven-dest-addr)#protocol udp
Router(config-model-driven-dest-addr)#commit
```

The UDP destination is shown as `Active` irrespective of the state of the collector because UDP is connectionless.

Model-driven Telemetry with UDP is not suitable for a busy network. There is no retry if a message is dropped by the network before it reaches the collector.

Example: Destination Group for gRPC Dial-out

Note gRPC is supported in only 64-bit platforms.

gRPC protocol supports TLS and model-driven telemetry uses TLS to dial-out by default. The certificate must be copied to `/misc/config/grpc/dialout/`. To by-pass the TLS option, use `protocol grpc no-tls`.

The following is an example of a certificate to which the server certificate is connected:

```
RP/0/RP0/CPU0:ios#run

Wed Aug 24 05:05:46.206 UTC
[xr-vm_node0_RP0_CPU0:~]$ls -l /misc/config/grpc/dialout/
total 4
-rw-r--r-- 1 root root 4017 Aug 19 19:17 dialout.pem
[xr-vm_node0_RP0_CPU0:~]$
```

The CN (CommonName) used in the certificate must be configured as `protocol grpc tls-hostname <>`.

The following example shows a destination group `DGroup2` created for gRPC dial-out configuration with key-value GPB encoding, and with tls disabled:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#destination-group DGroup2
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 grpc no-tls
Router(config-model-driven-dest-addr)#commit
```

The following example shows a destination group `DGroup2` created for gRPC dial-out configuration with key-value GPB encoding, and with tls hostname:

```
Configuration with tls-hostname:
Router(config)#telemetry model-driven
```

```
Router(config-model-driven)#destination-group DGroup2
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 grpc tls-hostname hostname.com
Router(config-model-driven-dest-addr)#commit
```

Note If only the **protocol grpc** is configured without tls option, tls is enabled by default and tls-hostname defaults to the IP address of the destination.

What to Do Next:

Create a sensor group.

Create a Sensor Group

The sensor-group specifies a list of YANG models that are to be streamed.

1. Identify the sensor path for XR YANG model.
2. Create a sensor group.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group <group-name>
Router(config-model-driven-snsr-grp)# sensor-path <XR YANG model>
Router(config-model-driven-snsr-grp)# commit
```

Example: Sensor Group for Dial-out

Note gRPC is supported in only 64-bit platforms.

The following example shows a sensor group `SGroup1` created for dial-out configuration with the YANG model for optics controller:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group SGroup1
Router(config-model-driven-snsr-grp)# sensor-path
Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info
Router(config-model-driven-snsr-grp)# commit
```

What to Do Next:

Create a subscription.

Create a Subscription

The subscription associates a destination-group with a sensor-group and sets the streaming method - cadence-based or event-based telemetry.

A source interface in the subscription group specifies the interface that will be used for establishing the session to stream data to the destination. If both VRF and source interface are configured, the source interface must be in the same VRF as the one specified under destination group for the session to be established.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription <subscription-name>
Router(config-model-driven-subs)#sensor-group-id <sensor-group> sample-interval <interval>

Router(config-model-driven-subs)#destination-id <destination-group>
Router(config-model-driven-subs)#source-interface <source-interface>
Router(config-mdt-subscription)#commit
```

Example: Subscription for Cadence-based Dial-out Configuration

The following example shows a subscription `Sub1` that is created to associate the sensor-group and destination-group, and configure an interval of 30 seconds to stream data:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription Sub1
Router(config-model-driven-subs)#sensor-group-id SGroup1 sample-interval 30000
Router(config-model-driven-subs)#destination-id DGroup1
Router(config-mdt-subscription)# commit
```

Example: Configure Event-driven Telemetry for Optics Controller and Performance Monitoring

```
telemetry model-driven
 destination-group 1
  address family ipv4 <ip-address> port <port-number>
   encoding self-describing-gpb
   protocol grpc no-tls
  !
 !
 sensor-group 1
  sensor-path
Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info
 !
 sensor-group 2
  sensor-path Cisco-IOS-XR-pmengine-oper:performance-management-history/global/periodic/
  optics-history/optics-port-histories/optics-port-history/optics-second30-history
 !
 subscription 1
  sensor-group-id 1 sample-interval 0
  sensor-group-id 2 sample-interval 0
  destination-id 1
!
```

Example: Subscription for Event-based Dial-out Configuration

The following example shows a subscription `Sub1` that is created to associate the sensor-group and destination-group, and configure event-based method to stream data:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription Sub1
Router(config-model-driven-subs)#sensor-group-id SGroup1 sample-interval 0
Router(config-model-driven-subs)#destination-id DGroup1
Router(config-mdt-subscription)# commit
```

What to Do Next:

Validate the configuration.

Validate Dial-out Configuration

Use the following command to verify that you have correctly configured the router for dial-out.

```
Router#show telemetry model-driven subscription <subscription-group-name>
```

Example: Validation for TCP Dial-out

```
Router#show telemetry model-driven subscription Sub1
Thu Jul 21 15:42:27.751 UTC
Subscription:  Sub1                     State: ACTIVE
-------------
  Sensor groups:
  Id                Interval(ms)        State
  SGroup1           30000               Resolved

  Destination Groups:
  Id                Encoding            Transport   State   Port    IP
  DGroup1           self-describing-gpb tcp         Active  5432    172.0.0.0
```

Example: Validation for gRPC Dial-out

Note gRPC is supported in only 64-bit platforms.

```
Router#show telemetry model-driven subscription Sub2
Thu Jul 21 21:14:08.636 UTC
Subscription:  Sub2                     State: ACTIVE
-------------
  Sensor groups:
  Id                Interval(ms)        State
  SGroup2           30000               Resolved

  Destination Groups:
  Id                Encoding            Transport   State   Port    IP
  DGroup2           self-describing-gpb grpc        ACTIVE      57500   172.0.0.0
```

The telemetry data starts steaming out of the router to the destination.

Example: Configure model-driven telemetry with different sensor groups

```
RP/0/RP0/CPU0:ios#sh run telemetry model-driven

Wed Aug 24 04:49:19.309 UTC

telemetry model-driven
 destination-group 1
  address family ipv4 10.1.1.1 port 1111
   protocol grpc
  !
 !

 destination-group 2
  address family ipv4 10.2.2.2 port 2222
  !
 !
```

```
 destination-group test
  address family ipv4 172.0.0.0 port 8801
   encoding self-describing-gpb
   protocol grpc no-tls
  !
  address family ipv4 172.0.0.0 port 8901
   encoding self-describing-gpb
   protocol grpc tls-hostname chkpt1.com
  !
 !

 sensor-group 1
  sensor-path
Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info
 !

 sensor-group mdt
  sensor-path Cisco-IOS-XR-pmengine-oper:performance-management-history/global/periodic/
  optics-history/optics-port-histories/optics-port-history/optics-second30-history
 !

 sensor-group generic
  sensor-path Cisco-IOS-XR-pmengine-oper:performance-management-history/global/periodic/
  optics-history/optics-port-histories/optics-port-history/optics-minute15-history
 !

 sensor-group if-oper
  sensor-path Cisco-IOS-XR-pmengine-oper:performance-management-history/global/periodic/
  optics-history/optics-port-histories/optics-port-history/optics-hour24-history
 !

 subscription mdt
  sensor-group-id mdt sample-interval 10000
 !

 subscription generic
  sensor-group-id generic sample-interval 10000
 !

 subscription if-oper
  sensor-group-id if-oper sample-interval 10000
  destination-id test
 !
!
```

A sample output from the destination with TLS certificate `chkpt1.com`:

```
RP/0/RP0/CPU0:ios#sh telemetry model-driven dest

Wed Aug 24 04:49:25.030 UTC
  Group Id        IP              Port    Encoding            Transport   State

  ------------------------------------------------------------------------------
  1               10.1.1.1         1111    none                grpc        ACTIVE

      TLS:10.1.1.1
  2               10.2.2.2         2222    none                grpc        ACTIVE

      TLS:10.2.2.2
  test            172.0.0.0   8801    self-describing-gpb grpc        Active
  test            172.0.0.0   8901    self-describing-gpb grpc        Active
      TLS:chkpt1.com
```

A sample output from the subscription:

```
RP/0/RP0/CPU0:ios#sh telemetry model-driven subscription

Wed Aug 24 04:49:48.002 UTC
Subscription:  mdt                     State: ACTIVE
-------------
  Sensor groups:
  Id                Interval(ms)        State
  mdt               10000               Resolved

Subscription:  generic                 State: ACTIVE

-------------
  Sensor groups:
  Id                Interval(ms)        State
  generic           10000               Resolved

Subscription:  if-oper                 State: ACTIVE
-------------
  Sensor groups:
  Id                Interval(ms)        State
  if-oper           10000               Resolved

  Destination Groups:
  Id                Encoding            Transport   State   Port    IP
  test              self-describing-gpb grpc        ACTIVE  8801    172.0.0.0

    No TLS :

  test              self-describing-gpb grpc        Active  8901    172.0.0.0
    TLS :             chkpt1.com


RP/0/RP0/CPU0:ios#sh telemetry model-driven subscription if-oper

Wed Aug 24 04:50:02.295 UTC
Subscription:  if-oper
-------------
  State:       ACTIVE
  Sensor groups:
  Id: if-oper
    Sample Interval:      10000 ms
    Sensor Path:
Cisco-IOS-XR-pmengine-oper:performance-management-history/global/periodic/
    optics-history/optics-port-histories/optics-port-history/optics-hour24-history
    Sensor Path State:    Resolved

  Destination Groups:
  Group Id: test
    Destination IP:       172.0.0.0
    Destination Port:     8801
    Encoding:             self-describing-gpb
    Transport:            grpc
    State:                ACTIVE
    No TLS
    Destination IP:       172.0.0.0
    Destination Port:     8901
    Encoding:             self-describing-gpb
    Transport:            grpc
    State:                ACTIVE
    TLS :                 chkpt1.com
    Total bytes sent:     120703
    Total packets sent:   11
    Last Sent time:       2016-08-24 04:49:53.52169253 +0000

  Collection Groups:
```

```
  ------------------
    Id: 1
    Sample Interval:      10000 ms
    Encoding:             self-describing-gpb
    Num of collection:    11
    Collection time:      Min:    69 ms Max:    82 ms
    Total time:           Min:    69 ms Avg:    76 ms Max:    83 ms
    Total Deferred:       0
    Total Send Errors:    0
    Total Send Drops:     0
    Total Other Errors:   0
    Last Collection Start:2016-08-24 04:49:53.52086253 +0000
    Last Collection End:  2016-08-24 04:49:53.52169253 +0000
    Sensor Path:
Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info
```

Configure Dial-in Mode

In a dial-in mode, the destination initiates a session to the router and subscribes to data to be streamed.

Note Dial-in mode is supported over gRPC in only 64-bit platforms.

For more information about dial-in mode, see *Dial-in Mode*.

The process to configure a dial-in mode involves these tasks:

- Enable gRPC
- Create a sensor group
- Create a subscription
- Validate the configuration

Enable gRPC

Configure the gRPC server on the router to accept incoming connections from the collector.

1. Enable gRPC over an HTTP/2 connection.

   ```
   Router# configure
   Router (config)# grpc
   ```

2. Enable access to a specified port number.

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

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

3. In the configuration mode, set the session parameters.

   ```
   Router (config)# grpc{ address-family | dscp | max-request-per-user | max-request-total
    | max-streams | max-streams-per-user | no-tls | service-layer | tls-cipher | tls-mutual
    | tls-trustpoint | vrf }
   ```

   where:

- **address-family:** set the address family identifier type
- **dscp:** set QoS marking DSCP on transmitted gRPC
- **max-request-per-user:** set the maximum concurrent requests per user
- **max-request-total:** set the maximum concurrent requests in total
- **max-streams:** set the maximum number of concurrent gRPC requests. The maximum subscription limit is 128 requests. The default is 32 requests
- **max-streams-per-user:** set the maximum concurrent gRPC requests for each user. The maximum subscription limit is 128 requests. The default is 32 requests
- **no-tls:** disable transport layer security (TLS). The TLS is enabled by default.
- **service-layer:** enable the grpc service layer configuration
- **tls-cipher:** enable the gRPC TLS cipher suites
- **tls-mutual:** set the mutual authentication
- **tls-trustpoint:** configure trustpoint
- **server-vrf:** enable server vrf

4. Commit the configuration.

```
Router(config-grpc)#commit
```

The following example shows the output of `show grpc` command. The sample output displays the gRPC configuration when TLS is enabled on the router.

```
Router#show grpc

Address family              : ipv4
Port                        : 57300
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-aes256-gcm-sha384
                            : ecdhe-ecdsa-aes256-gcm-sha384
                            : ecdhe-rsa-aes128-sha
                            : ecdhe-ecdsa-aes128-sha
                            : ecdhe-rsa-aes256-sha
                            : ecdhe-ecdsa-aes256-sha
                            : aes128-gcm-sha256
```

```
                              : aes256-gcm-sha384
                              : aes128-sha
                              : aes256-sha
  Operational disable         : none
```

What to Do Next:

Create a sensor group.

Create a Sensor Group

The sensor group specifies a list of YANG models that are to be streamed.

1. Identify the sensor path for XR YANG model.
2. Create a sensor group.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group <group-name>
Router(config-model-driven-snsr-grp)# sensor-path <XR YANG model>
Router(config-model-driven-snsr-grp)# commit
```

Example: Sensor Group for gRPC Dial-in

The following example shows a sensor group `SGroup3` created for gRPC dial-in configuration with the YANG model for interfaces:

```
Router(config)#telemetry model-driven
Router(config-model-driven)#sensor-group SGroup3
Router(config-model-driven-snsr-grp)# sensor-path openconfig-interfaces:interfaces/interface

Router(config-model-driven-snsr-grp)# commit
```

What to Do Next:

Create a subscription.

Create a Subscription

The subscription associates a sensor-group with a streaming interval. The collector requests the subscription to the sensor paths when it establishes a connection with the router.

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription <subscription-name>
Router(config-model-driven-subs)#sensor-group-id <sensor-group> sample-interval <interval>

Router(config-model-driven-subs)#destination-id <destination-group>
Router(config-mdt-subscription)#commit
```

Example: Subscription for gRPC Dial-in

The following example shows a subscription `Sub3` that is created to associate the sensor-group with an interval of 30 seconds to stream data:

```
Router(config)telemetry model-driven
Router(config-model-driven)#subscription Sub3
```

```
Router(config-model-driven-subs)#sensor-group-id SGroup3 sample-interval 30000
Router(config-mdt-subscription)#commit
```

What to Do Next:

Validate the configuration.

Validate Dial-in Configuration

Use the following command to verify that you have correctly configured the router for gRPC dial-in.

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

Example: Validation for gRPC Dial-in

```
RP/0/RP0/CPU0:SunC#show telemetry model-driven subscription Sub3
Thu Jul 21 21:32:45.365 UTC
Subscription:  Sub3
-------------
  State:       ACTIVE
  Sensor groups:
  Id: SGroup3
    Sample Interval:      30000 ms
    Sensor Path:          openconfig-interfaces:interfaces/interface
    Sensor Path State:    Resolved

  Destination Groups:
  Group Id: DialIn_1002
    Destination IP:       172.30.8.4
    Destination Port:     44841
    Encoding:             self-describing-gpb
    Transport:            dialin
    State:                Active
    Total bytes sent:     13909
    Total packets sent:   14
    Last Sent time:       2016-07-21 21:32:25.231964501 +0000

  Collection Groups:
  ------------------
    Id: 2
    Sample Interval:      30000 ms
    Encoding:             self-describing-gpb
    Num of collection:    7
    Collection time:      Min:    32 ms Max:    39 ms
    Total time:           Min:    34 ms Avg:    37 ms Max:    40 ms
    Total Deferred:       0
    Total Send Errors:    0
    Total Send Drops:     0
    Total Other Errors:   0
    Last Collection Start:2016-07-21 21:32:25.231930501 +0000
    Last Collection End:  2016-07-21 21:32:25.231969501 +0000
    Sensor Path:          openconfig-interfaces:interfaces/interface
```

Event-driven Telemetry for Terminal-device Models

In R6.5.2, event-driven telemetry is supported for terminal-device models. When an alarm is received, the alarm is immediately sent through the telemetry system. The event-driven telemetry is enabled by setting the sample interval value to 0 in the subscription configuration.

Example: Configure Event-driven Telemetry for Terminal-device Models

RP/0/RP0/CPU0:ios# show running-config telemetry model-driven

```
Wed Sep 19 13:57:41.418 IST
telemetry model-driven
destination-group pipeline_test
  address-family ipv4 198.51.100.3 port 5890
   encoding self-describing-gpb
   protocol tcp
  !
!

 sensor-group gkl_30seconds
  sensor-path openconfig-system:system
!
subscription gkl_30seconds
  sensor-group-id gkl_30seconds sample-interval 0
  destination-id pipeline_test
!
!
```

sensor-path openconfig-system:system means open config sensor path (global).

sample-interval 0 means telemetry is performed instantly for alarm occurrence and clearance.

Verify the Resolution of Sensor Path

Use the following show command to verify whether the sensor path is resolved.

RP/0/RP0/CPU0:ios# show telemetry model-driven subscription gkl_30s$

```
Wed Sep 26 09:59:48.326 IST
Subscription:  gkl_30seconds
-------------
  State:       Paused
  Sensor groups:
  Id: gkl_30seconds
    Sample Interval:      0 ms
    Sensor Path:          openconfig-system:system
    Sensor Path State:    Resolved

  Destination Groups:
  Group Id: pipeline_test
    Destination IP:       10.77.132.122
    Destination Port:     5900
    Encoding:             self-describing-gpb
    Transport:            tcp
    State:                NA
    No TLS

  Collection Groups:
  ------------------
```

```
Id: 1
Sample Interval:      0 ms
Encoding:             self-describing-gpb
Num of collection:    24
Collection time:      Min:     7 ms Max:    15 ms
Total time:           Min:     1 ms Avg:     5 ms Max:    15 ms
Total Deferred:       9
Total Send Errors:    0
Total Send Drops:     0
Total Other Errors:   0
No data Instances:    0
Last Collection Start:2018-09-24 18:22:36.1991344829 +0530
Last Collection End:  2018-09-25 12:39:56.3406424389 +0530
Sensor Path:          openconfig-system:system
```

Streaming Event-Driven Telemetry for Online Insertion and Removal of Pluggables

Table 1: Feature History

Feature Name	Release	Description
Event Driven Telemetry Support for Online Insertion and Removal (OIR) of Pluggables	Cisco IOS XR Release 7.8.1	A new sensor path in the OpenConfig model type is introduced to support EDT in NCS 1004 during OIR of the pluggables. It triggers telemetry data such as form factor, SONET-SDH compliance code, FEC corrected bits during removal, and state, channel data during insertion of the NCS 1004 chassis. This telemetry data helps you to track the pluggables present in the NCS 1004 chassis.

Event-driven telemetry in NCS 1004 streams operational data that are related to each lane that is configured for pluggables when OIR of pluggables occurs. In this section, the output examples show the operational data for pluggables that are configured with a single lane and four lanes. The `openconfig-platform-transceiver:transceiver` sensor path in the OpenConfig RPC model provides telemetry data of NCS 1004 pluggables that are removed or added in the NCS 1004 chassis.

Enabling EDT for OIR of Pluggables

To enable the event-driven telemetry for the OIR of pluggables, perform the following steps in order.

1. Use the `no no-tls` command in the gRPC configuration mode to enable the event-driven telemetry.
2. Run the subscription configuration file and input file together in the following format. Use the following command in your local machine to which you want to stream the event-driven telemetry data for pluggables that you remove or add.

```
<local-file-path>/<client-file> -a <IPv4-address>:<gRPC-portnumber> -insecure
-insecure_username <username> -insecure_password <password> -<encoding> "$(cat
<subscription-config file)" -dt <display-type-string>
```

Table 2: Attribute Description

Attribute	Data Type	Description
<local-file-path>	file path	Local file path of the client and subscription configuration file in your machine.
<client-file>	String	Name of the client file to enable EDT
<IPv4-address>	Decimal	IPv4 address of the NCS 1004 chassis
<gRPC-portnumber>	Integer	Port number of the gRPC port
<username>	String	Name of the admin user
<password>	Alphanumeric	User password to access the NCS 1004 chassis.
<encoding>	String	Type of the encoding format
<subscription-config file>	String	Name of the configuration file to enable EDT subscription
<display-type-string>	Character	Format of the output display

The following sample command executes the subscription file to stream the telemetry data for the OIR of the pluggables.

```
/ws/achakali-bgl/bh_final/bh_devtest/bh_auto/bh_automation/generated_files/gnmi_cli_latest
 -a 10.127.60.146:57400 -insecure -insecure_username test2 -insecure_password cisco123
-proto "$(cat transceiver_input)" -dt p
```

Subscription Configuration File

The following sample configuration file is based on gNMI specifications. It uses the openconfig sensor path and enables the EDT in NCS 1004 for the OIR of pluggables.

Note Event-driven telemetry is enabled by setting the sample interval value to **0** and mode to **ON_CHANGE** in the subscription configuration.

```
subscribe: <
    prefix: <
    >
    subscription: <
      path: <
            elem: <
                name: "openconfig-platform:components"
            >
            elem: <
                name: "component/openconfig-platform-transceiver:transceiver"
            >
          >
```

```
      mode: ON_CHANGE
      sample_interval: 0
    >
      mode: STREAM
      encoding: PROTO
>
```

The sensor path **component/openconfig-platform-transceiver:transceiver** enables the streaming of transceiver data when the pluggable is inserted or removed. The encoding format **proto** displays the streamed telemetry data in the **.proto** format. The mode **STREAM** enables the stream subscription for the set of sensory paths. For more information on the **gNMI** specifications, refer to gRPC Network Management Interface (gNMI).

Verify the Sensor Path

Use the following command to verify whether the event-driven telemetry sensor path for the OIR of pluggables in NCS 1004 is enabled.

```
RP/0/RP0/CPU0:ios#show telemetry model-driven internal subscription
```

The following output shows the **component/openconfig-platform-transceiver:transceiver** sensor path is active.

```
Fri Oct 21 15:32:28.785 IST
Subscription:  GNMI__10083376335112435231
-------------
  State:       ACTIVE
  Sensor groups:
  Id: GNMI__10083376335112435231_0
    Sample Interval:      0 ms
    Heartbeat Interval:   NA
    Sensor Path:
openconfig-platform:components/component/openconfig-platform-transceiver:transceiver
    Sensor Path State:    Resolved

  Destination Groups:
  Group Id: GNMI_1001
    Destination IP:       198.51.100.3
    Destination Port:     60058
    Encoding:             gnmi-proto
    Transport:            dialin
    State:                Active
    TLS :                 True
    Total bytes sent:     309553
    Total packets sent:   179
    Last Sent time:       2022-10-21 15:32:15.2304030650 +0530

  Collection Groups:
  ------------------
    Id: 1
    Sample Interval:      0 ms
    Heartbeat Interval:   NA
    Heartbeat always:     False
    Encoding:             gnmi-proto
    Num of collection:    1
    Incremental updates:  0
    Collection time:      Min:   709 ms Max:   709 ms
    Total time:           Min:   716 ms Avg:   716 ms Max:   716 ms
    Total Deferred:       1
    Total Send Errors:    0
    Total Send Drops:     0
    Total Other Errors:   0
```

```
    No data Instances:    0
    Last Collection Start:2022-10-21 15:32:14.2303313823 +0530
    Last Collection End:  2022-10-21 15:32:15.2304030650 +0530
    Sensor Path:
openconfig-platform:components/component/openconfig-platform-transceiver:transceiver

      Sysdb Path:
/oper/overlays/gl/oc_transceiver/openconfig-platform/components/component__list_S/*/transceiver/physical-channels/channel__list_u__bag_overlay_oc_transceiver_channel/*

      Count:          1 Method: FINDDATA Min: 709 ms Avg: 709 ms Max: 709 ms
      Item Count:     132 Status: Eventing Active
      Missed Collections:0  send bytes: 286891 packets: 130 dropped bytes: 0
      Missed Heartbeats: 0  Filtered Item Count: 0
                      success         errors          deferred/drops
      Gets            0               0
      List            0               0
      Datalist        0               0
      Finddata        3               0
      GetBulk         0               0
      Encode                          0               1
      Send                            0               0

    Id: 2
    Sample Interval:      0 ms
    Heartbeat Interval:   NA
    Heartbeat always:     False
    Encoding:             gnmi-proto
    Num of collection:    1
    Incremental updates:  0
    Collection time:      Min:   691 ms Max:   691 ms
    Total time:           Min:   694 ms Avg:   694 ms Max:   694 ms
    Total Deferred:       0
    Total Send Errors:    0
    Total Send Drops:     0
    Total Other Errors:   0
    No data Instances:    0
    Last Collection Start:2022-10-21 15:32:14.2302824953 +0530
    Last Collection End:  2022-10-21 15:32:15.2303519103 +0530
    Sensor Path:
openconfig-platform:components/component/openconfig-platform-transceiver:transceiver

      Sysdb Path:
/oper/overlays/gl/oc_transceiver/openconfig-platform/components/component__list_S/*/transceiver/state__bag_overlay_oc_transceiver_state

      Count:          1 Method: FINDDATA Min: 691 ms Avg: 691 ms Max: 691 ms
      Item Count:     49 Status: Eventing Active
      Missed Collections:0  send bytes: 22662 packets: 48 dropped bytes: 0
      Missed Heartbeats: 0  Filtered Item Count: 0
                      success         errors          deferred/drops
      Gets            0               0
      List            0               0
      Datalist        0               0
      Finddata        2               0
      GetBulk         0               0
      Encode                          0               0
      Send                            0               0

RP/0/RP0/CPU0:ios#
```

EDT Output for the OIR of Pluggables

Output for Inserted Pluggable

The following example shows the telemetry data for the addition of a single lane FR-S pluggable optic transceiver in port 13 of a 1.2T card in slot 1 in **.proto** format. The following output is the same for a single lane LR-S pluggable optic transceiver.

```
update: <
    path: <
      elem: <
        name: "state"
      >
      elem: <
        name: "present"
      >
    >
    val: <
      string_val: "PRESENT"
    >
  >
>

update: <
  timestamp: 1667367012319000000
  prefix: <
    origin: "openconfig-platform"
    elem: <
      name: "components"
    >
    elem: <
      name: "component"
      key: <
        key: "name"
        value: "Optics0_1_0_13"
      >
    >
    elem: <
      name: "openconfig-platform-transceiver:transceiver"
    >
  >
  update: <
    path: <
      elem: <
        name: "physical-channels"
      >
      elem: <
        name: "channel"
        key: <
          key: "index"
          value: "1"
        >
      >
      elem: <
        name: "state"
      >
      elem: <
        name: "index"
      >
    >
    val: <
      uint_val: 1
    >
  >
>
```

The following table describes the highlighted parameters in the preceding example.

Table 3: Parameters Description

Parameters	Description
name: "openconfig-platform-transceiver:transceiver"	Subscribed sensor path
update:	Addition of the transceiver pluggable
string_val: "PRESENT"	Indicates the availability of the pluggable
value: "Optics0_1_0_13"	FR pluggable inserted in slot 1 port 13 of 1.2T card

Output for Removed Pluggable

The following example shows the telemetry data for the removal of a single lane LR-S pluggable optic transceiver in port 13 of a 1.2T card in slot 1 in **.proto** format. The following output is the same for a single lane FR-S pluggable optic transceiver.

```
update: <
  timestamp: 1667367004302000000
  prefix: <
    origin: "openconfig-platform"
    elem: <
      name: "components"
    >
    elem: <
      name: "component"
      key: <
        key: "name"
        value: "Optics0_1_0_13"
      >
    >
    elem: <
      name: "openconfig-platform-transceiver:transceiver"
    >
  >
  delete: <
    elem: <
      name: "state"
    >
    elem: <
      name: "fec-mode"
    >
  >

.
.
output snipped
.
.
  delete: <
    elem: <
      name: "state"
    >
    elem: <
      name: "fec-uncorrectable-words"
    >
  >
  delete: <
    elem: <
      name: "state"
```

```
    >
    elem: <
      name: "fec-corrected-bits"
    >
  >
>

update: <
  timestamp: 1667367004303000000
  prefix: <
    origin: "openconfig-platform"
    elem: <
      name: "components"
    >
    elem: <
      name: "component"
      key: <
        key: "name"
        value: "Optics0_1_0_13"
      >
    >
    elem: <
      name: "openconfig-platform-transceiver:transceiver"
    >
  >
  delete: <
    elem: <
      name: "physical-channels"
    >
    elem: <
      name: "channel"
      key: <
        key: "index"
        value: "1"
      >
    >
    elem: <
      name: "state"
    >
    elem: <
      name: "index"
    >
  >
  delete: <
    elem: <
      name: "physical-channels"
    >
    elem: <
      name: "channel"
      key: <
        key: "index"
        value: "1"
      >
    >
    elem: <
      name: "state"
    >
    elem: <
      name: "description"
    >
  >
  delete: <
    elem: <
      name: "physical-channels"
```

```
    >
    elem: <
      name: "channel"
      key: <
        key: "index"
        value: "1"
      >
    >
    elem: <
      name: "state"
    >
>
```

The following table describes the highlighted parameters in the preceding example.

Table 4: Parameters Description

Parameters	Description
name: "openconfig-platform-transceiver:transceiver"	Subscribed sensor path
delete:	Removal of the transceiver pluggable
value: "Optics0_1_0_13"	LR pluggable removed in slot 1 port 13 of 1.2T card
key: < key: "index" value: "1"	Indicates the deleted lane number

The following example shows the telemetry data for the removal of a four-lane LR4-S pluggable optic transceiver in port 5 of a 1.2T card in slot 0 in **.proto** format.

```
update: <
  timestamp: 1667367249665000000
  prefix: <
    origin: "openconfig-platform"
    elem: <
      name: "components"
    >
    elem: <
      name: "component"
      key: <
        key: "name"
        value: "Optics0_0_0_5"
      >
    >
    elem: <
      name: "openconfig-platform-transceiver:transceiver"
    >
  >
  delete: <
    elem: <
      name: "state"
    >
    elem: <
      name: "fec-mode"
    >
  >
.
.
output snipped
```

```
.
.
  delete: <
    elem: <
      name: "state"
    >
    elem: <
      name: "fec-uncorrectable-words"
    >
  >
  delete: <
    elem: <
      name: "state"
    >
    elem: <
      name: "fec-corrected-bits"
    >
  >
>

update: <
  timestamp: 1667367249666000000
  prefix: <
    origin: "openconfig-platform"
    elem: <
      name: "components"
    >
    elem: <
      name: "component"
      key: <
        key: "name"
        value: "Optics0_0_0_5"
      >
    >
    elem: <
      name: "openconfig-platform-transceiver:transceiver"
    >
  >
  delete: <
    elem: <
      name: "physical-channels"
    >
    elem: <
      name: "channel"
      key: <
        key: "index"
        value: "1"
      >
    >
    elem: <
      name: "state"
    >
    elem: <
      name: "index"
    >
  >
.
.
.output snipped
.
.
>

update: <
```

```
  timestamp: 1667367249667000000
  prefix: <
    origin: "openconfig-platform"
    elem: <
      name: "components"
    >
    elem: <
      name: "component"
      key: <
        key: "name"
        value: "Optics0_0_0_5"
      >
    >
    elem: <
      name: "openconfig-platform-transceiver:transceiver"
    >
  >
  delete: <
    elem: <
      name: "physical-channels"
    >
    elem: <
      name: "channel"
      key: <
        key: "index"
        value: "2"
      >
    >
    elem: <
      name: "state"
    >
    elem: <
      name: "index"
    >
  >
.
.
output snipped
.
.
>

update: <
  timestamp: 1667367249669000000
  prefix: <
    origin: "openconfig-platform"
    elem: <
      name: "components"
    >
    elem: <
      name: "component"
      key: <
        key: "name"
        value: "Optics0_0_0_5"
      >
    >
    elem: <
      name: "openconfig-platform-transceiver:transceiver"
    >
  >
  delete: <
    elem: <
      name: "physical-channels"
    >
```

```
    elem: <
      name: "channel"
      key: <
        key: "index"
        value: "3"
      >
    >
    elem: <
      name: "state"
    >
    elem: <
      name: "index"
    >
.
.
output snipped
.
.
  >
>

update: <
  timestamp: 1667367249670000000
  prefix: <
    origin: "openconfig-platform"
    elem: <
      name: "components"
    >
    elem: <
      name: "component"
      key: <
        key: "name"
        value: "Optics0_0_0_5"
      >
    >
    elem: <
      name: "openconfig-platform-transceiver:transceiver"
    >
  >
  delete: <
    elem: <
      name: "physical-channels"
    >
    elem: <
      name: "channel"
      key: <
        key: "index"
        value: "4"
      >
    >
    elem: <
      name: "state"
    >
    elem: <
      name: "index"
    >
  >
.
.
output snipped
.
.
>
```

The following table describes the highlighted parameters in the preceding example.

Table 5: Parameters Description

Parameters	Description
name: "openconfig-platform-transceiver:transceiver"	Subscribed sensor path
delete:	Removal of the transceiver pluggable
value: "Optics0_0_0_5"	LR4 pluggable removed in slot 0 port 5 of 1.2T card
key: < key: "index" value: "4"	Indicates the deleted lane number 4

gRPC Network Management Interface

gRPC Network Management Interface is an interface for a network management system to interact with a network element.

gNMI Services

- Get - Used by the client to retrieve configuration data on the target.
- Set - Used by the client to modify configuration data of the target.
- Telemetry - Used by the client to control subscriptions to the data on the target.

Example for GET:

Syntax:

```
$./gnmi_cli -get --address=mrstn-5502-2.cisco.com:57344 \
  -proto "$(cat test.proto)" \
  -with_user_pass \
  -insecure \
  -ca_crt=ca.cert \
  -client_crt=ems.pem \
  -client_key=ems.key \
  -timeout=5s
```

Example for SET:

Syntax:

```
$ ./gnmi_cli -set --address=mrstn-5502-2.cisco.com:57344 \
  -proto "$(cat test.proto)" \
  -with_user_pass \
  -insecure \
  -ca_crt=ca.cert \
  -client_crt=ems.pem \
  -client_key=ems.key \
  -timeout=5s
```

Example for Subscribe:

Syntax:

```
$ ./gnmi_cli --address=mrstn-5502-2.cisco.com:57344 \
  -proto "$(cat test.proto)" \
  -with_user_pass \
  -insecure \
  -ca_crt=ca.cert \
  -client_crt=ems.pem \
  -client_key=ems.key \
  -timeout=5s
  -display_type string (g, group, s, single, p, proto). (default "group")
```

Subscription Mode

Subscription Mode is the mode of the subscription, specifying how the target must return values in a subscription.

Modes of the subscription:

- STREAM = 0
- ONCE = 1
- POLL = 2

ONCE Subscriptions: A subscription operating in the ONCE mode acts as a single request/response channel. The target creates the relevant update messages, transmits them, and subsequently closes the RPC.

STREAM Subscriptions: Stream subscriptions are long-lived subscriptions which continue to transmit updates relating to the set of paths that are covered within the subscription indefinitely.

2s Telemetry Based on GNMI Subscribe

gRPC Network Management Interface (GNMI) is a network management protocol used for configuration management and telemetry. gNMI provides the mechanism to install, manipulate, and delete the configuration of network devices, and to view operational data. The content provided through gNMI can be modeled using YANG.

Typically, GNMI client is configured to receive telemetry reports for every 30 seconds. The user can configure GNMI client with a sample interval of two seconds. However, the system cannot manage this delay between two collections. Hence, a characterization has been done to evaluate the actual system performance.

The characterization started to identify the maximum system load corresponding to the following scenario:

- The node is configured as a section protection node (EDFA, PSM, and EDFA modules on the three slots).
- Both the EDFA modules are configurated with grid mode=50Ghz.

  The grid-mode configuration creates up to 96 additional OTS-OCH controllers for each ots 0/slot/0/0 and ots 0/slot/0/1. The grid-mode 50GHz configuration adds 96 * 2 (number of slots with EDFA module for section protection) * 2 (bidirectional ports having OTS-OCH controllers for each EDFA module) = 384 controllers to the system.

Measurements have been performed for maximum load and for no_grid mode.

The following sensor paths are supported for telemetry testing in NCS 1001. GNMI client is configured for the following sensor paths to receive telemetry reports for every two seconds.

Sensor Path	Description
Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info	Provides the data of all the OTS and OTS-OCH controllers for the system.
Cisco-IOS-XR-controller-optics-oper:optics-oper/optics-ports/optics-port/optics-info Cisco-IOS-XR-alarm-server-oper:alarm-short/alarm-additional-location-short/active Characterization is performed with line-rx and com-rx disconnected on both the EDFA modules. OTS-OCH controller is in maintenance state by default.	Provides the controller information of OTS and OTS-OCH controllers and active alarms.

The above sensor paths must belong to the same sensor group for which the minimum subscription interval is measured. Other sensor paths can belong to other sensor groups with subscriptions greater than 30 seconds.

The characterization performed found a value for which a sample interval for telemetry is suggested to 10 seconds for maximum load, whereas subscription time is 5 seconds for grid_mode=no grid.

gNMI Heartbeat Interval

Table 6: Feature History

Feature Name	Release Information	Feature Description
gNMI Heartbeat Interval	Cisco IOS XR Release 7.3.2	The gNMI Heartbeat Interval feature allows you to send ON_CHANGE subscription data for each heartbeat interval regardless of change in value. This feature enables you to enhance the network management system.

The gNMI heartbeat interval must be specified along with **ON_CHANGE** subscription. In this case, the value of the data items must be resent once for each heartbeat interval regardless of change in value.

Limitation

Enabling the gNMI heartbeat interval using Open Config is not supported.

Examples

The following example uses the openconfig sensor path and enables the gNMI heartbeat interval value of one hour in nanoseconds.

```
subscribe: <
    prefix: <
    >
    subscription: <
      path: <
            origin: "openconfig-system"
            elem: <
                name: "system"
```

```
            >
            elem: <
                name: "alarms"
            >
          >
      mode: ON_CHANGE
      heartbeat_interval: 3600000000000
    >
        mode: STREAM
        encoding: PROTO
>
```

The following example shows the enabled gNMI heartbeat interval.

```
RP/0/RP0/CPU0:ios#show run telemetry model-driven subscription sub-1
Thu Jun 17 08:41:52.400 UTC
telemetry model-driven
subscription sub-1
  sensor-group-id group1 sample-interval 0
  sensor-group-id group1 heartbeat interval 3600000000000
  sensor-group-id group1 heartbeat always
!
!
```

The **interval** attribute sends subscription data for each heartbeat interval when no events have occurred within the interval. The **always** attribute sends subscription data for each heartbeat interval even if events have occurred within the interval. The **sample-interval** attribute is enabled only with event-driven telemetry. This attribute value must be set to 0 to enable event-driven telemetry.