

Enhancements to Streaming Telemetry

This section provides an overview of the enhancements made to streaming telemetry data.

- Hardware Timestamp, on page 2
- Enhanced Syslog Notifications for Unresolved Line Card Forwarding Paths, on page 4
- Target-Defined Mode for Cached Generic Counters Data, on page 5
- gNMI Dial-Out via Tunnel Service, on page 8
- Stream Telemetry Data about PBR Decapsulation Statistics, on page 10
- Stream Telemetry Data for ACL, on page 12
- Stream Telemetry Data for BGP FlowSpec, on page 17
- View Internal TCAM Resource Utilization for Ingress Hybrid ACL, on page 22
- Stream telemetry for IPv4 and IPv6 data on network interfaces, on page 23
- Timestamp in nano seconds, on page 31

Hardware Timestamp

Table 1: Feature History Table

Feature Name	Release Information	Description
Enhancements to Hardware Timestamp	Release 7.3.4	Telemetry messages carry a timestamp per interface to indicate the time when data is collected from the hardware. With this feature, the support for hardware timestamp is extended to MPLS Traffic Engineering (MPLS TE) counters, Segment Routing for Traffic Engineering (SR-TE) interface counters, protocol statistics, and bundle protocol counters. The interface counters in the following paths are enhanced for hardware timestamp: • Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters • Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters • openconfig-network-instance:network-instances/network-instance/mpls/lsps/constrained-path/tunnels • openconfig-interfaces:interfaces/interface
Hardware Timestamp	Release 7.3.1	Whenever periodic statistics are streamed, the collector reads the data from its internal cache, instead of fetching the data from the hardware. When the data is read from the cache, the rate at which data is processed shows spikes because the timestamp from the collector is off by several seconds. With hardware timestamping, the inconsistencies that are observed when reading data from the cache file is removed.

Whenever periodic stats are streamed, the collector reads the stats from its internal cache, instead of fetching the stats from the hardware. When the data is read from the sensor paths of Stats manager cache, the rate calculation shows spikes. This behavior is due to the timestamp from the collector that is off by several seconds.

Therefore, timestamp of some other collector takes precedence because timestamps of collectors are not in synchronization with the current timestamp. This is observed when there are multiple collectors providing stats updates for the same interface.

The YANG data model for Stats manager `Cisco-IOS-XR-infra-statsd-oper.yang` is enhanced to enable the collector to read periodic stats data from the router using hardware timestamp.

The hardware timestamp is taken into account when a primary collector (for generic or proto stats) provides stats updates from the hardware to the Stats manager. With hardware timestamping in rate computation while streaming periodic stats, the spikes due to the timestamp issue is resolved.

The hardware timestamp is updated only when the collector attempts to read the counters from hardware. Else, the value remains `0`. The latest stats can be streamed at a minimum cadence of 10 seconds and periodic stats at a cadence of 30 seconds. The support is available only for physical interfaces and subinterfaces, and bundle interface and subinterfaces.

When there is no traffic flow on protocols for an interface, the hardware timestamp for the protocols is published as 0. This is due to non-synchronized timestamps sent by the collector for protocols in traffic as compared to non-traffic scenarios.

A non-zero value is published for protocols that have stats published by a primary collector for both traffic and non-traffic scenarios.



Note The hardware timestamp is supported only for primary collectors. When the hardware has no update, the timestamp will be same. However generic counters are computed for primary and non-primary collectors. The non-primary collectors show the latest stats, but not the timestamp.

When the counters are cleared for an interface using **clear counters interface** command, all counter-related data including the timestamps for the interface is cleared. After all counter values are cleared and set to `0`, the last data time is updated only when there is a request for it from a collector. For example, last data time gets updated from a collector:

```
Router#:Aug  7 09:01:08.471 UTC: statsd_manager_l[168]: Updated last data time for ifhandle
 0x02000408,
stats type 2 from collector with node 0x100, JID 250, last data time 1596790868.
INPUT: last 4294967295 updated 1596469986. OUTPUT: last 4294967295 updated 1596469986
```

All other counter values and hardware timestamp are updated when the counters are fetched from the hardware. In this case, all counters including the hardware timestamp is 0:

```
{"node_id_str":"MGBL_MTB_5504","subscription_id_str":"app_TEST_200000001",
"encoding_path":"Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters",
"collection_id":"7848",
"collection_start_time":"1596790879567",
"msg_timestamp":"1596790879571","data_json":
[{"timestamp":"1596790879570","keys":[{"interface-name":"FortyGigE0/1/0/11"}],
"content":{"packets-received":"0","bytes-received":"0","packets-sent":"0",
"bytes-sent":"0","multicast-packets-received":"0","broadcast-packets-received":"0",
"multicast-packets-sent":"0","broadcast-packets-sent":"0","output-drops":0,"output-queue-drops":0,
"input-drops":0,"input-queue-drops":0,"runt-packets-received":0,"giant-packets-received":0,
"throttled-packets-received":0,"parity-packets-received":0,"unknown-protocol-packets-received":0,
"input-errors":0,"crc-errors":0,"input-overruns":0,"framing-errors-received":0,"input-ignored-packets":0,
"input-aborts":0,"output-errors":0,"output-underruns":0,"output-buffer-failures":0,"output-buffers-swapped-out":0,
"applique":0,"resets":0,"carrier-transitions":0,"availability-flag":0,
"last-data-time":"1596790868","hardware-timestamp":"0",
"seconds-since-last-clear-counters":15,"last-discontinuity-time":1596469946,"seconds-since-packet-received":0,
"seconds-since-packet-sent":0}}],"collection_end_time":"1596790879571"}
```

Enhanced Syslog Notifications for Unresolved Line Card Forwarding Paths

Table 2: Feature History Table

Feature Name	Release Information	Description
Enhanced Syslog Notifications for Unresolved Line Card Forwarding Paths	Release 7.3.3	This feature notifies you of Line Card and Route Processor paths not resolving in the Forwarding Information Base. Both Model-Driven Telemetry (MDT) and Event Driven Telemetry (EDT) notifications are supported. In earlier releases, notifications for route processors were supported. This feature provides for improved diagnostics.

Telemetry now supports syslog notification from line cards. This is in addition to the existing notification support from route processors. You will be notified of line card and route processor paths not resolving in the Forwarding Information Base (FIB), through MDT and EDT notifications.

MDT is configured for cadence-based telemetry, while EDT is configured for event-based notification. Notifications are generated only when the device goes into error or OOR state, and during device recovery. Errors and OOR are tracked for a device as a whole, and not for individual nodes. The IPv4 Error, IPv6 Error, IPv4 OOR, and IPv6 OOR telemetry notifications are supported.

The following notification is an example of IPv4 error state, if a line card and route processor paths do not resolve in the FIB:

```
GPB(common) Message
[5.13.9.177:38418(PE1)/Cisco-IOS-XR-fib-common-oper:oc-aft-l3/protocol/ipv4/error/state msg
 len: 168]
{
    "Source": "5.13.9.177:38418",
    "Telemetry": {
        "node_id_str": "PE1",
        "subscription_id_str": "Sub2",
       "encoding_path": "Cisco-IOS-XR-fib-common-oper:oc-aft-l3/protocol/ipv4/error/state",

        "collection_id": 243,
        "collection_start_time": 1637858634881,
        "msg_timestamp": 1637858634881,
        "collection_end_time": 1637858634883
    },
    "Rows": [
        {
            "Timestamp": 1637858634882,
            "Keys": null,
            "Content": {
                "is-in-error-state": "true"
            }
        }
```

```
    ]
}
```

Note The parameters denote`"Content": {"is-in-error-state": "true"}` that the system is in error state.

The following notification is an example of IPv4 OOR, if a line card and route processor are in OOR state:

```
GPB(common) Message
[5.13.9.177:50146(PE1)/Cisco-IOS-XR-fib-common-oper:oc-aft-l3/protocol/ipv4/oor/state msg
len: 163]
{
    "Source": "5.13.9.177:50146",
    "Telemetry": {
        "node_id_str": "PE1",
        "subscription_id_str": "Sub1",
        "encoding_path": "Cisco-IOS-XR-fib-common-oper:oc-aft-l3/protocol/ipv4/oor/state",
        "collection_id": 11,
        "collection_start_time": 1637815892624,
        "msg_timestamp": 1637815892624,
        "collection_end_time": 1637815892626
    },
    "Rows": [
        {
            "Timestamp": 1637815892625,
            "Keys": null,
            "Content": {
                "is-in-oor-state": "true"
            }
        }
    ]
}
```

Note The parameters denote`"Content": {"is-in-oor-state": "true"}` that the system is in OOR state.

Target-Defined Mode for Cached Generic Counters Data

Table 3: Feature History Table

Feature Name	Release Information	Description
Target-Defined Mode for Cached Generic Counters Data		This feature streams telemetry data for cached generic counters using a TARGET_DEFINED subscription. This subscription ensures that any change to the cache streams the latest data to the collector as an event-driven telemetry notification. This feature introduces support for the following sensor path: `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters`

Streaming telemetry pushes the subscribed data from the router to one or more collectors. The telemetry infrastructure retrieves the data from the system database when you send a subscription request. Based on the subscription request or the telemetry configuration the cached generic counters data can be retrieved periodically based on the sample-interval. Data, such as interface statistics, is cached and refreshed at certain intervals. The `TARGET_DEFINED` subscription mode can be used to retrieve data when the cache gets updated, and is not based on a timer.

The application can register as a data producer with the telemetry library and the SysdB paths it supports. One of the data producers, Statsd, uses the library with a `TARGET_DEFINED` subscription mode. As part of this mode, the producer registers the sensor paths. The statistics infrastructure streams the incremental updates for statsd cache sensor path `Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/generic-counters`. With this path in the subscription, whenever cache is updated, the statsd application pushes the updates to the telemetry daemon. The daemon sends these incremental updates to the collector. The cache updates are pushed for physical interfaces, physical subinterfaces, bundle interfaces, and bundle subinterfaces. You can subscribe to the sensor path for the cached generic counters with `TARGET_DEFINED` mode instead of the sensor path for the latest generic counters (`Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/latest/generic-counters`) to reduce the system load.

Configure the router to stream telemetry data from cache for generic counters using the following instructions:

Create a `TARGET_DEFINED` subscription mode for cached generic counters using one of the two options:

- **Option 1:** gRPC Network Management Interface (gNMI) subscribe request

```
{
    "name": "SubscribeRequest",
    "subscribe": {
        "prefix": {"origin":
                "Cisco-IOS-XR-infra-statsd-oper"

             },
         "mode": "STREAM", "encoding": "PROTO", "updates_only": "false",
        "subscription": [
         { "path": {"elem": [ {"name":"infra-statistics"},
                              {"name":"interfaces"},
                              {"name":"interface"},
                              {"name":"cache"},
                              {"name":"generic-counters"}
                            ]
             },
           "mode":   "TARGET_DEFINED"
         }
                     ]
    }
}
```

- **Option 2:** Model-driven telemetry configuration for non-gNMI requests

```
Router(config)#telemetry model-driven
Router(config-model-driven)#subscription sub1
Router(config-model-driven-subs)#sensor-group-id grp1 mode target-defined
Router(config-model-driven-subs)#source-interface Interface1
Router(config-model-driven-subs)#commit
```

After the subscription is triggered, updates to the stats cache are monitored. The statsd application pushes the cached generic counters to the client (collector).

View the number of incremental updates for the sensor path.

```
Router#show telemetry model-driven subscription .*
Fri Nov 12 23:36:27.212 UTC
Subscription: GNMI__16489080148754121540
-------------
   Collection Groups:
    ------------------
      Id: 1
      Sample Interval:      0 ms   (Incremental Updates)
      Heartbeat Interval:   NA
      Heartbeat always:     False
      Encoding:             gnmi-proto
      Num of collection:    1
      Incremental updates:  12
      Collection time:      Min:     5 ms Max:     5 ms
      Total time:           Min:     6 ms Avg:     6 ms Max:     6 ms
      Total Deferred:       1
      Total Send Errors:    0
      Total Send Drops:     0
      Total Other Errors:   0
      No data Instances:    0
      Last Collection Start:2021-11-12
                 23:34:27.1362538876 +0000
      Last Collection End:  2021-11-12   23:34:27.1362545589
             +0000
      Sensor Path:          Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/
                            interface/cache/generic-counters
```

In this example, the incremental updates of `12` indicates that the cache is updated 12 times.

You can also retrieve the detailed operational data about the subscription using the following command. In this example, `statsd-target` is the subscription name.

```
Router#show telemetry model-driven subscription statsd-target internal
Fri Nov 12 08:51:16.728 UTC
Subscription: statsd-target
-------------
State: ACTIVE
Sensor groups:
Id: statsd
Sample Interval: 0 ms (Incremental Updates)
Heartbeat Interval: NA
Sensor Path: Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface/cache/
             generic-counters
Sensor Path State: Resolved

Destination Groups:
Group Id: statsd-target
Destination IP: 192.0.2.1
Destination Port: 56000
Encoding: json
Transport: grpc
State: Active
TLS : False
Total bytes sent: 623656
Total packets sent: 13
Last Sent time: 2021-08-16 08:51:15.1304821089 +0000

Collection Groups:
------------------
Id: 2
Sample Interval: 0 ms (Incremental Updates)
Heartbeat Interval: NA
Heartbeat always: False
```

```
Encoding: json
Num of collection: 1
Incremental updates: 3
Collection time: Min: 94 ms Max: 94 ms
Total time: Min: 100 ms Avg: 100 ms Max: 100 ms
Total Deferred: 0
Total Send Errors: 0
Total Send Drops: 0
Total Other Errors: 0
No data Instances: 0
Last Collection Start:2021-08-16 08:51:04.1293895665 +0000
Last Collection End: 2021-08-16 08:51:04.1293996284 +0000
```

The sample interval of `0` indicates that the data is streamed whenever an event occurs. Here, the event represents the updates to the cache state.

Related Commands:

- **show tech telemetry model-driven**
- **show running-config telemetry model-driven**
- **show telemetry producers trace** *producer name* **info**
- **show telemetry producers trace** *producer name* **err**

gNMI Dial-Out via Tunnel Service

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

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

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

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

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

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

Procedure

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

Example:

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

A default route is automatically gained in the Linux shell.

Step 2 Configure the gNMI tunnel service on the router.

Example:

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

Where—

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

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

Example:

```
Router#show run grpc
Wed Nov 24 19:37:21.015 UTC
grpc
  port 57500
  no-tls
  tunnel
    destination 5.0.0.2 port 59510
      target Target-1
      source-interface GigabitEthernet0/0/0/1
    !
    destination 2002::1:2 port 59510
       source-interface GigabitEthernet0/0/0/0
    !
    destination 192.0.0.1 port 59500
    !
    destination 192.0.0.1 port 59600
    !
  !
!
```

Step 4 View the status of tunnel destination.

Example:

```
Router#show grpc tunnel sessions
Wed Nov 24 19:41:38.863 UTC
 5.0.0.2:59510
 Target:           Target-1
```

```
Status:           Not connected
Error:            Source Interface is down
Source interface: GigabitEthernet0/0/0/1
Source address:   5.0.0.1
Source VRF:       default

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

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

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

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

Step 6 Run the collector.

Stream Telemetry Data about PBR Decapsulation Statistics

You can stream telemetry data about PBR decapsulation statistics for GRE and GUE encapsulation protocols that deliver packets using IPv4 or IPv6. The encapsulated data has source and destination address that must match with the source and destination address in the classmap. Both encapsulation and decapsulation interfaces collect statistics periodically. The statistics can be displayed on demand using **show policy-map type pbr [vrf vrf-name] address-family ipv4/ipv6 statistics** command. For more information on PBR-based decapsulation, see *Interface and Hardware Component Configuration Guide for Cisco NCS 5500 Series Routers*.

With this release, the decapsulation statistics can be displayed using `Cisco-IOS-XR-infra-policymgr-oper.yang` data model and telemetry data. You can stream telemetry data from the sensor path:

`Cisco-IOS-XR-infra-policymgr-oper:policy-manager/global/policy-map/policy-map-types/policy-map-type/vrf-table/vrf/afi-table/afi/stats`

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

Procedure

Step 1 Check the running configuration to view the configured PBR per VRF.

Example:

```
Router#show running-config
Building configuration...
!! IOS XR Configuration 0.0.0
!!
vrf vrf1
 address-family ipv4 unicast
 !
 address-family ipv6 multicast
 !
!
netconf-yang agent
 ssh
!
!
class-map type traffic match-all cmap1
 match protocol gre
 match source-address ipv4 161.0.1.1 255.255.255.255
 match destination-address ipv4 161.2.1.1 255.255.255.255
 end-class-map
!
policy-map type pbr gre-policy
 class type traffic cmap1
  decapsulate gre
 !
 class type traffic class-default
 !
 end-policy-map
!
interface GigabitEthernet0/0/0/1
 vrf vrf1
 ipv4 address 2.2.2.2 255.255.255.0
 shutdown
!
vrf-policy
 vrf vrf1 address-family ipv4 policy type pbr input gre-policy
!
end
```

Step 2 View the output of the VRF statistics.

Example:

```
Router#show policy-map type pbr vrf vrf1 addr-family ipv4 statistics

VRF Name:       vrf1
Policy-Name:    gre-policy
Policy Type:    pbr
Addr Family:    IPv4

Class:     cmap1
     Classification statistics      (packets/bytes)
       Matched             :       13387587/1713611136
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :       13387587/1713611136

Class:     class-default
     Classification statistics      (packets/bytes)
       Matched             :              0/0
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :              0/0
```

After you have verified that the statistics are displayed correctly, stream telemetry data and check the streamed data at the collector. For more information about collectors, see *Operate on Telemetry Data for In-depth Analysis of the Network* section in the Monitor CPU Utilization Using Telemetry Data to Plan Network Infrastructure chapter.

```
ios.0/0/CPU0/ $ mdt_exec -s Cisco-IOS-XR-infra-policymgr-oper:policy-manager
/global/policy-map/policy-map-types/policy-map-type/vrf-table/vrf/afi-table/afi/stats -c 100
{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001","encoding_path":
"Cisco-IOS-XR-infra-policymgr-oper:policy-manager/global/policy-map/policy-map-types/policy-map-type
/vrf-table/vrf/afi-table/afi/stats","collection_id":"1","collection_start_time":"1601361558157",
"msg_timestamp":"1601361559179","data_json":[{"timestamp":"1601361559178","keys":[{"type":"ipv6"},
{"vrf-name":"vrf_gue_ipv4"},{"type":"ipv4"}],"content":{"pmap-name":"gre-policy","vrf-name":
"vrf1","appln-type":2,"addr-family":1,"rc":0,"plmgr-vrf-stats":[{"pmap-name":"gre-policy",
"cmap-stats-arr":[{"cmap-name":"cmap1","matched-bytes":"1713611136","matched-packets":"13387587",
"transmit-bytes":"1713611136","transmit-packets":"13387587"}]}]}}],
"collection_end_time":"1601361559183"}
---------------------------------  snipped for brevity ------------------------------------
```

Stream Telemetry Data for ACL

Table 4: Feature History Table

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

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

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

You can stream ACL telemetry data from the following XPaths:

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

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

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

SUMMARY STEPS

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

DETAILED STEPS

Procedure

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

Example:

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

Example:

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

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

Example:

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

Example:

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

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

After you have verified that the statistics are displayed correctly, stream telemetry data and check the streamed data at the collector. For more information about Model-Driven Telemetry collectors, see Establish a Model-Driven Telemetry Session from a Router to a Collector.

Example:

MDT of ACL IPv4 statistics

```
Router# run mdt_exec -s Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/
access-list-manager/accesses/access/access-list-sequences/access-list-sequence -c 30000
Enter any key to exit...
Request datatree:
    filter
        ipv4-acl-and-prefix-list (ka)
            access-list-manager
                accesses
                    access
                        access-list-sequences
                            access-list-sequence
Sub_id 200000001, flag 0, len 0
Sub_id 200000001, flag 4, len 6739
--------
{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001",
"encoding_path":"Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/
accesses/access/access-list-sequences/access-list-sequence","collection_id":"1",
"collection_start_time":"1655427578624","msg_timestamp":"1655427578632",
"data_json":[{"timestamp":"1655427578629","keys":[{"access-list-name":"tel_test"},
{"sequence-number":10}],"content":{"item-type":"normal","sequence":10,"grant":"permit",
"protocol-operator":0,"protocol":512,"protocol2":0,"source-address":"0.0.0.0","source-address-mask":"255.255.255.255"
--------
"fragment-offset1":0,"fragment-offset2":0,"set-ttl":65535,"fragment-flags":0,"police":{"police-value":0,
"police-unit":"pps","police-peak-value":0,"police-peak-unit":"pps"},"priority":"acl-priority-unspec",
"is-icmpon":false}}],"collection_end_time":"1655427578633"}
--------
```

Example:

MDT of ACL IPv6 statistics:

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

You can apply filter on ACL name as followed:

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

Stream Telemetry Data for ACL Byte Counters

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

- Stream entire ACL data:

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

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

- Stream specific ACL data:

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

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

- Stream specific sequence of ACL data:

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

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

View the telemetry subscription with the ACL sensor paths.

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

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

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

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

View the byte counters configured for ACL statistics.

The following example shows the statistics for IPv4 address family:

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

The following example shows the statistics for IPv6 address family:

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

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

MDT of ACL IPv4 statistics:

```
Router#run mdt_exec -s
"Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/access-list-manager/accesses/access
[access-list-name=="v4"]/access-list-sequences/access-list-sequence[sequence-number=10]"
-c 300000
Thu Mar 30 06:19:06.698 UTC
Enter any key to exit...
 Sub_id 200000001, flag 0, len 0
 Sub_id 200000001, flag 4, len 5452
--------
{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001","encoding_path":"Cisco-IOS-XR-ipv4-acl-oper:ipv4-acl-and-prefix-list/
access-list-manager/accesses/access/access-list-sequences/access-list-sequence","collection_id":"1","collection_start_time":"1663309156852",
"msg_timestamp":"1663309156862","data_json":[{"timestamp":"1663309156860","keys":[{"access-list-name":"v4"},{"sequence-number":10}],
"content":{"item-type":"normal","sequence":10,"grant":"permit","protocol-operator":0,"protocol":512,"protocol2":0,"source-address":
"0.0.0.0","source-address-mask":"255.255.255.255","destination-address":"2.2.0.0","destination-address-mask":"0.0.255.255","source-operator":
"none","source-port1":0,"source-port2":0,"destination-operator":"none","destination-port1":0,"destination-port2":0,"log-option":
"log-none","capture":false,"dscp-present":true,"dscp":10,"dscp2":0,"dscp-operator":0,"dscp-bitmask":255,"precedence-present":false,
"precedence":0,"tcp-flags-operator":"match-none","tcp-flags":0,"tcp-flags-mask":0,"fragments":0,"packet-length-operator":"none",
"packet-length1":0,"packet-length2":0,"ttl-operator":"none","ttl1":0,"ttl2":0,"no-stats":false,"hits":"3","hardware-hits":"11035388",
"police-hits":"0","police-hits-dropped":"0","byte-hits":"1368388112","byte-police-hits":"0","byte-police-hits-dropped":"0","is-icmp-off":
false,"qos-group":65535,"next-hop-type":"nexthop-none","next-hop-info":[{"next-hop":"0.0.0.0","status":"not-present","at-status":
"unknown","is-acl-next-hop-exist":false},{"next-hop":"0.0.0.0","status":"not-present","at-status":"unknown","is-acl-next-hop-exist":
false},{"next-hop":"0.0.0.0","status":"not-present","at-status":"unknown","is-acl-next-hop-exist":false}],"dynamic":false,"acl-name":
"v4","sequence-str":"10","fragment-offset-operator":"none","fragment-offset1":0,"fragment-offset2":0,"set-ttl":65535,"fragment-flags":
0,"police":{"police-value":0,"police-unit":"pps","police-peak-value":0,"police-peak-unit":"pps"},"priority":"acl-priority-unspec",
"is-icmpon":false}}
------------
```

MDT of ACL IPv6 statistics:

```
Router#run mdt_exec -s
"Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/access-list-manager/accesses/
access[access-list-name="v6"]/access-list-sequences/access-list-sequence[sequence-number=10]"
 -c 300000
Thu Mar 30 06:19:41.464 UTC
Enter any key to exit...
 Sub_id 200000001, flag 0, len 0
 Sub_id 200000001, flag 4, len 6247
--------
{"node_id_str":"ios","subscription_id_str":"app_TEST_200000001","encoding_path":"Cisco-IOS-XR-ipv6-acl-oper:ipv6-acl-and-prefix-list/
access-list-manager/accesses/access/access-list-sequences/access-list-sequence","collection_id":"3","collection_start_time":"1663309191611",
"msg_timestamp":"1663309191620","data_json":[{"timestamp":"1663309191618","keys":[{"access-list-name":"v6"},{"sequence-number":10}],
"content":{"is-ace-type":"normal","is-ace-sequence-number":10,"is-packet-allow-or-deny":"permit","is-protocol-operator":"none",
"is-ipv6-protocol-type":513,"is-ipv6-protocol2-type":0,"is-source-address-in-numbers":"::","is-source-address-prefix-length":0,
"source-mask":"::","is-destination-address-in-numbers":"2222::","is-destination-address-prefix-length":64,"destination-mask":
"ffff:ffff:ffff:ffff::","is-source-operator":"none","is-source-port1":0,"is-source-port2":0,"is-destination-operator":"none",
"is-destination-port1":0,"is-destination-port2":0,"is-log-option":"log-none","is-tcp-bits-operator":"match-none","is-tcp-bits":0,
"is-tcp-bits-mask":0,"is-dscp-present":1,"dscp-operator":0,"is-dscp-valu":10,"is-dscp-valu2":0,"dscp-bitmask":255,
"is-precedence-present":0,"is-precedence-value":0,"is-header-matches":0,"is-packet-length-operator":"none","is-packet-length-start":
```

0,"is-packet-length-end":0,"is-time-to-live-operator":"none","is-time-to-live-start":0,"is-time-to-live-end":0,"no-stats":0,"hits":"2","hardware-hits":"11035388","police-hits":"0","police-hits-dropped":"0","byte-hits":"1368388112","byte-police-hits":"0","byte-police-hits-dropped":"0","capture":0,"undetermined-transport":0,"is-icmp-message-off":0,"qos-group":65535,"next-hop-type":"nexthop-none","next-hop-info":[{"next-hop":"::","vrf-name":"","track-name":"","status":"not-present","at-status":"unknown","acl-nh-exist":0},{"next-hop":"::","vrf-name":"","track-name":"","status":"not-present","at-status":"unknown","acl-nh-exist":0},{"next-hop":"::","vrf-name":"","track-name":"","status":"not-present","at-status":"unknown","acl-nh-exist":0}],"is-flow-id":4294967295,"acl-name":"v6","sequence-str":"10","set-ttl":65535,"police":{"police-value":0,"police-unit":"pps","police-peak-value":0,"police-peak-unit":"pps"},"priority":"acl-priority-unspec","fragment-flags":0,"is-icmp-message-on":0}}

Stream Telemetry Data for BGP FlowSpec

Table 5: Feature History Table

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

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

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

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

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

SUMMARY STEPS

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

DETAILED STEPS

Procedure

Step 1 Check the configuration of the BGP FlowSpec.

Example:

```
Router# show running-config
Client config:
router bgp 100
nsr
bgp router-id 2.2.2.1
address-family ipv4 unicast
!
address-family vpnv4 unicast
!
address-family ipv6 unicast
!
address-family vpnv6 unicast
!
address-family ipv4 flowspec
!
address-family ipv6 flowspec
!
address-family vpnv4 flowspec
!
address-family vpnv6 flowspec
!
neighbor 1.1.1.1
  remote-as 100
  update-source Loopback1
  address-family ipv4 unicast
  !
  address-family vpnv4 unicast
  !
  address-family ipv4 flowspec
  !
  address-family vpnv4 flowspec
  !
!
neighbor 1.1.1.2
 remote-as 100
  update-source Loopback2
  address-family ipv6 unicast
  !
  address-family vpnv6 unicast
  !
  address-family ipv6 flowspec
  !
  address-family vpnv6 flowspec
  !
!
!
flowspec
local-install interface-all
address-family ipv4
  local-install interface-all
  service-policy type pbr redirect
!
!
```

```
end

class-map type traffic match-all c1
match protocol sctp
 end-class-map
!
!
class-map type traffic match-all c2
match protocol udp
 end-class-map
!
class-map type traffic match-all c3
match dscp 3
 end-class-map
!
class-map type traffic match-all c1_6
match dscp af11
 end-class-map
!
class-map type traffic match-all c2_6
match dscp 20
 end-class-map
!
policy-map type pbr p1
class type traffic c1
  drop
!
 class type traffic c2
  drop
!
 class type traffic c3
  drop
!
 class type traffic class-default
 !
 end-policy-map
!
policy-map type pbr p1_6
class type traffic c1_6
  set dscp af21
!
 class type traffic c2_6
  set dscp af22
!
 class type traffic class-default
 !
 end-policy-map
!
router bgp 100
nsr
bgp router-id 1.1.1.1
address-family ipv4 unicast
!
address-family vpnv4 unicast
!
address-family ipv6 unicast
!
address-family vpnv6 unicast
!
address-family ipv4 flowspec
!
address-family ipv6 flowspec
!
address-family vpnv4 flowspec
```

```
!
address-family vpnv6 flowspec
!
neighbor 2.2.2.1
  remote-as 100
  update-source Loopback1
  address-family ipv4 unicast
  !
  address-family vpnv4 unicast
  !
  address-family ipv4 flowspec
  !
  address-family vpnv4 flowspec
  !
!
neighbor 2.2.2.2
  remote-as 100
  update-source Loopback2
  address-family ipv6 unicast
  !
  address-family vpnv6 unicast
  !
  address-family ipv6 flowspec
  !
  address-family vpnv6 flowspec
  !
!
!
flowspec
address-family ipv4
  service-policy type pbr p1
!
address-family ipv6
  service-policy type pbr p1_6
!
!
```

Step 2 View BGP FlowSpec statistics for IPv4 and IPv6.

Example:

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

```
    Actions      :Traffic-rate: 0 bps  (bgp.1)
   Statistics                       (packets/bytes)
      Matched             :                  0/0
      Transmitted         :                  0/0
      Dropped             :                  0/0


AFI: IPv6
  Flow           :DSCP:=10
Flowspec Rule:
    Matches:
      DSCP                :                  10
    Actions      :DSCP: af21  (bgp.1)
    Statistics                        (packets/bytes)
      Matched             :                  0/0
      Transmitted         :                  0/0
      Dropped             :                  0/0
  Flow           :DSCP:=20
Flowspec Rule:
    Matches:
      DSCP                :                  20
    Actions      :DSCP: af22  (bgp.1)
    Statistics                        (packets/bytes)
      Matched             :                  0/0
      Transmitted         :                  0/0
      Dropped
```

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

After you have verified that the statistics are displayed correctly, stream telemetry data and check the streamed data at the collector. For more information about Model-Driven Telemetry collectors, see Establish a Model-Driven Telemetry Session from a Router to a Collector.

Example:

MDT of BGP FlowSpec statistics

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

View Internal TCAM Resource Utilization for Ingress Hybrid ACL

Table 6: Feature History Table

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

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

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

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

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

View TCAM Usage

Display data using CLI

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

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

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

Display data using Streaming Telemetry

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

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

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

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

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

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

Configuration Example

Specify the subset of the data that you want to stream from the router using sensor paths. The **sensor path** represents the path in the hierarchy of a YANG data model.

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

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

Stream telemetry for IPv4 and IPv6 data on network interfaces

Streaming telemetry for IPv4 and IPv6 data on network interfaces is a method of continuously collecting and transmitting real-time data using standardized OpenConfig data models and gNMI sensor paths. This approach:

- enables real-time monitoring and reporting of IPv4 and IPv6 operational states and configuration changes, and

- improves network management with standardized data models for seamless multi-vendor compatibility,

Table 7: Feature History Table

Feature Name	Release Information	Description
Stream telemetry for IPv4 and IPv6 data on network interfaces	Release 25.2.1	Introduced in this release on: NCS 5500 fixed port routers; NCS 5700 fixed port routers; NCS 5500 modular routers (NCS 5500 line cards; NCS 5700 line cards [Mode: Compatibility; Native] You can now enhance network reliability and resource optimization by monitoring IPv4 and IPv6 performance and operational states across platforms. This ensures consistent management, proactive troubleshooting, and optimization in multi-vendor environments using openconfig-if-ip.yang data models and telemetry-enabled sensor paths.

Streaming telemetry data for openconfig data model through gNMI supports monitoring of various data points, include:

- operational status,
- configuration changes, and
- performance metrics.

gNMI sensor paths to stream IPv4 and IPv6 telemetry data

You can stream telemetry data from these gNMI sensor paths using On Change subscription mode or at a cadence of 30 seconds or higher (with a scale of 2000 interfaces). For more details on gNMI subscription, see the GitHub repository.

- openconfig-interfaces/interface/state

IPv4 sub-interface level:

- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv4/state
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv4/addresses
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv4/neighbor
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv4/proxy-arp

IPv6 sub-interface level:

- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/state
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/addresses
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/router-advertisement
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/openconfig-if-ip-ext:autoconf
- openconfig-interfaces:interfaces/interface/subinterfaces/subinterface/openconfig-if-ip:ipv6/neighbor

Verify telemetry data for IPv4 and IPv6 on network interfaces

You can verify the telemetry data for IPv4 and IPv6 data on network interfaces.

Procedure

Verify the output of the interface IP statistics.

Example:

This example shows IPv4 state information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.1:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv4/state'
{
    "source": "192.168.2.1:17933",
    "timestamp": 1746203252773061780,
    "time": "2025-05-02T12:27:32.77306178-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv4/state",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv4/state": {
            "counters": {
              "in-multicast-octets": "0",
              "in-multicast-pkts": "0",
              "in-octets": "0",
              "in-pkts": "0",
              "out-multicast-octets": "0",
              "out-multicast-pkts": "0",
              "out-octets": "0",
              "out-pkts": "0"
            },
            "dhcp-client": false,
            "mtu": 1500
          }
        }
      }
    ]
  }
]
```

Example:

This example shows IPv4 address information.

```
auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.2:17933 --username xxxxx --password xxxxxxxx
--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv4/addresses'
[
  {
    "source": "192.168.2.2:17933",
    "timestamp": 1746203286230765971,
    "time": "2025-05-02T12:28:06.230765971-04:00",
    "updates": [
      {
        "Path":
```

```
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv4/addresses",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv4/addresses": {
            "address": [
              {
                "config": {
                  "ip": "192.168.10.1",
                  "prefix-length": 24,
                  "type": "PRIMARY"
                },
                "ip": "192.168.20.1",
                "state": {
                  "ip": "192.168.20.1",
                  "origin": "STATIC",
                  "prefix-length": 24,
                  "type": "PRIMARY"
                }
              }
            ]
          }
        }
      }
    ]
  }
]
```


Example:

This example shows IPv4 neighbor information.


```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.3:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv4/neighbors'
[
  {
    "source": "192.168.2.3:17933 ",
    "timestamp": 1746203327097683095,
    "time": "2025-05-02T12:28:47.097683095-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv4/neighbors",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv4/neighbors": {
            "neighbor": [
              {
                "ip": "192.168.20.1",
                "state": {
                  "ip": "192.168.20.1",
                  "link-layer-address": "78:bf:38:b6:66:08",
                  "origin": "OTHER"
                }
              },
              {
                "ip": "192.168.20.2",
                "state": {
                  "ip": "192.168.20.2",
                  "link-layer-address": "00:00:00:00:00:00",
                  "origin": "OTHER"
                }
              }
```

```
            ]
          }
        }
      }
    ]
  }
]
```

Example:

This example shows IPv4 proxy-arp information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.4:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv4/proxy-arp'
[
  {
    "source": "192.168.2.4:17933",
    "timestamp": 1746203562540052546,
    "time": "2025-05-02T12:32:42.540052546-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv4/proxy-arp",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv4/proxy-arp": {
            "config": {
              "mode": "ALL"
            },
            "state": {
              "mode": "ALL"
            }
          }
        }
      }
    ]
  }
]
```

Example:

IPv6 state

This example shows IPv6 state information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.5:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/state'
[
  {
    "source": "192.168.2.5:17933",
    "timestamp": 1746202812260230182,
    "time": "2025-05-02T12:20:12.260230182-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv6/state",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv6/state": {
            "counters": {
              "in-multicast-octets": "0",
              "in-multicast-pkts": "0",
```

```
              "in-octets": "0",
              "in-pkts": "0",
              "out-multicast-octets": "0",
              "out-multicast-pkts": "0",
              "out-octets": "0",
              "out-pkts": "0"
            },
            "dup-addr-detect-transmits": 5,
            "enabled": true,
            "mtu": 1500
          }
        }
      }
    ]
  }
]
```

Example:

This example shows IPv6 address information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.6:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/addresses'
[
  {
    "source": "192.168.2.6:17933",
    "timestamp": 1746202852330541300,
    "time": "2025-05-02T12:20:52.3305413-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv6/addresses",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv6/addresses": {
            "address": [
              {
                "config": {
                  "ip": "10:10:3::1",
                  "prefix-length": 119,
                  "type": "GLOBAL_UNICAST"
                },
                "ip": "10:10:3::1",
                "state": {
                  "ip": "10:10:3::1",
                  "origin": "STATIC",
                  "prefix-length": 119,
                  "status": "PREFERRED",
                  "type": "GLOBAL_UNICAST"
                }
              },
              {
                "ip": "fe80::7abf:38ff:feb6:6608",
                "state": {
                  "ip": "fe80::7abf:38ff:feb6:6608",
                  "origin": "STATIC",
                  "prefix-length": 128,
                  "status": "PREFERRED",
                  "type": "LINK_LOCAL_UNICAST"
                }
              }
            ]
```

```
          }
        }
      }
    ]
  }
]
```

Example:

This example shows IPv6 neighbor information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.7:17933 --username xxxxx --password xxxxxxxx
 --
skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/neighbor'
[
  {
    "source": "192.168.2.7:17933",
    "timestamp": 1746202898868407604,
    "time": "2025-05-02T12:21:38.868407604-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]",

        "values": {
          "interfaces/interface/subinterfaces/subinterface": null
        }
      }
    ]
  }
]
```

Example:

This example shows IPv6 state duplicate addres transmits information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.8:17933 --username xxxxx --password xxxxxxxx

--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/state/dup-addr-detect-transmits'
[
  {
    "source": "192.168.2.8:17933",
    "timestamp": 1746202980834877546,
    "time": "2025-05-02T12:23:00.834877546-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv6/state/dup-addr-detect-transmits",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv6/state/dup-addr-detect-transmits": 5

        }
      }
    ]
  }
]
```

Example:

This example shows IPv6 router advertisement information.

```
/auto/tftpboot-ottawa/b4/bin/gnmic --address 192.168.2.9:17933 --username xxxxx --password xxxxxxxx
```

```
--skip-verify --encoding JSON_IETF get --path '/interfaces/interface[name=FourHundredGigE0/0/0/1]/
subinterfaces/subinterface[index=0]/ipv6/router-advertisement/'
[
  {
    "source": "192.168.2.9:17933",
    "timestamp": 1746202744369280518,
    "time": "2025-05-02T12:19:04.369280518-04:00",
    "updates": [
      {
        "Path":
"openconfig:interfaces/interface[name=FourHundredGigE0/0/0/1]/subinterfaces/subinterface[index=0]/ipv6/router-advertisement",

        "values": {
          "interfaces/interface/subinterfaces/subinterface/ipv6/router-advertisement": {
            "config": {
              "enable": true,
              "interval": 4,
              "lifetime": 9000,
              "managed": false,
              "other-config": true,
              "suppress": true
            },
            "prefixes": {
              "prefix": [
                {
                  "config": {
                    "disable-autoconfiguration": true,
                    "enable-onlink": true,
                    "preferred-lifetime": 5000,
                    "prefix": "300:0:2::/124",
                    "valid-lifetime": 6000
                  },
                  "prefix": "300:0:2::/124",
                  "state": {
                    "disable-autoconfiguration": true,
                    "enable-onlink": true,
                    "preferred-lifetime": 5000,
                    "prefix": "300:0:2::/124",
                    "valid-lifetime": 6000
                  }
                }
              ]
            },
            "state": {
              "enable": true,
              "interval": 4,
              "lifetime": 9000,
              "managed": false,
              "other-config": true,
              "suppress": true
            }
          }
        }
      }
    ]
  }
]
```

Timestamp in nano seconds

From Release 25.2.1, the telemetry messages for all sensor paths are populated with the `timestamp_nano` attribute. This is the time at which the data is collected from the underlying source, or the time that the message is generated, if provided by the underlying source. This is the number of nanoseconds since the Unix Epoch. For reference telemetry messages, see Github.

The primary benefit is the improved timestamp accuracy, which is now in nanoseconds rather than the milliseconds available earlier. Additionally, XR dial-in and dial-out telemetry includes the `timestamp_nano` field in the telemetry messages, ensuring more precise time tracking.

Timestamp in nano seconds