Hardware Timestamp

Table 1: Feature History Table

Feature Name	Release Information	Description
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"}
```

- Target-Defined Mode for Cached Generic Counters Data, on page 3
- Stream Telemetry Data about PBR Decapsulation Statistics, on page 5
- Stream telemetry for IPv4 and IPv6 data on network interfaces, on page 7
- Timestamp in nano seconds, on page 14

Target-Defined Mode for Cached Generic Counters Data

Table 2: 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**

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 540 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 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 3: Feature History Table

Feature Name	Release Information	Description
Stream telemetry for IPv4 and IPv6 data on network interfaces	Release 25.2.1	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.