APPENDIX E

IOS XR SNMP Best Practices

Simple Network Management Protocol (SNMP) is the most common network management protocol in the routing industry. This chapter describes best practices to be adopted by an Operations Support System (OSS) for optimized use of the IOS-XR SNMP protocol.

Note OSS platform tuning and Data Communication Network (DCN) considerations are outside the scope of this document.

Overview

The implementation of management infrastructure for IOS XR network elements usually makes use of SNMP as a first line tool, in particular for fault management and statistics reports.

This chapter provides guidelines on how to interface the SNMP Agent of the IOS XR network equipment in different ways. The goals are to:

- Prevent SNMP congestion
- Achieve a better response time
- Avoid losing traps

Timeouts and Retries

The access to SNMP tables or variables cannot be instantaneous, and given the UDP nature of the protocol, the OSS cannot rely on a fast response time. In addition, SNMP is a non confirmed protocol and therefore, the OSS has to be tuned to retry after a GET operation times out.

Timeouts

The timeout to be tuned on the SNMP management application depends on various factors:

- Response time of the Network Element (NE), which largely depends on the presence of data in cache vs. dynamic retrieve
- DCN delay, usually shorter than the above, but sometimes not negligible because of congestion
- Number of SNMP management applications polling the same NE

While a timeout of 1 second can be considered in a lab, in a real application at least 3.5 seconds is preferred. The best approach is a dynamic timeout, where the OSS automatically adjusts the timeout upon different poll retries. This approach, where present, optimizes response time and takes into account all of the above factors.

Where dynamic tuning is not available at OSS level, a row formula for a timeout can be the following:

```
N x TRT + DCNd
```

Where:

- N—Number of management applications (for example, SNMP Managers) polling an SNMP agent at the same time. Maximum recommended is 5
- RT—Regular timeout for a single management application. As stated above, 3.5 seconds is recommended
- DCNd—DCN delay for the given OSS or NE pair

Timeout Recommendations

1. Use dynamic timeout if available
2. Use a 3.5 second timeout if dynamic timeout is not available
3. Have no more than 5 management applications polling SNMP at the same time.

Retries

When a poll timeout expires, the typical OSS performs a certain number of retries before declaring an object as *unreachable*. For timeout fine tuning, the number of retries should also be tuned depending on the same factors. Dynamic adjustment is recommended when present, otherwise heuristic calculations need to be used.

As a general consideration, a retry should be done with the same SNMP request-id and the same IP source port. This takes the first available response and also helps the SNMP agent to drop multiple retries directed to the same object.

Retry Recommendations

1. Use dynamic retry if available
2. Calculate static retry based on factors listed, if dynamic retry is not available
3. On all retries, use the same SNMP request-id and IP source port

Tables

A typical MIB table is defined as SEQUENCE of SEQUENCE in MIB syntax, where each SEQUENCE contains a set of MIB objects. An instance of a SEQUENCE represents a row, and all instances of the MIB objects represent a column in a conceptual MIB table. The way in which a table is traversed significantly impacts the response time. Many considerations need to be examined when accessing a table and these depend on the table itself. First, the way in which the table is traversed may affect the response time, and secondly, the nature of the table itself suggests some good practices.

Accessing Tables

Data in a table can be retrieved by doing a sequence of SNMP GET-NEXT requests by one or more GET-BULK requests. Irrespective of the type of request, traversal in a table is essentially fetching the 'next' instance of an object. This is possible because MIB tables are expected to be sorted lexicographically based on one or more indices.

Each row in a conceptual MIB table contains attributes or statistics for a specific entity. What this entity represents is purely based on MIB definition. Most of the MIB implementations act as front end for a specific feature. The set of data that is required could be deep within a process or hardware counter that implements the actual feature. In a distributed environment, that data could even be in a line card. Various data pertaining to a specific entity is often kept together as a group and thus the cost of fetching this group is often the same as getting one element from this group—because of inter-process communication overhead (here the cost is in terms of CPU and time).

It is therefore much more efficient to access all objects for an instance or entity than accessing them one by one. This can be considered as row-by-row traversal. Achieving this with GET-NEXT or GET-BULK operation means specifying all required objects of a table in the same request. This approach is usually better, except in the case of sparse tables. For more information, see the "Sparse Tables" section on page E-218.

Table E-1 is an example of SNMP best practices when accessing a table.

Note In Table E-1 each line under 'Column-wise' represents the output of a single SNMP request and in the 'Row-wise' column not all of the objects are listed (for brevity each SNMP request includes only three objects).

Table E-1 Accessing the ifTable of IF-MIB

Column-wise	Row-wise
bash-2.03$ getmany -v2c 12.25.26.10 public ifTable	bash-2.03$ bash-2.03$ getmany -v2c 12.25.26.10 public ifDescr ifType ifAdminStatus

ifIndex.1 = 1
ifIndex.2 = 2
ifIndex.3 = 3
ifIndex.4 = 4
ifIndex.5 = 5
ifIndex.6 = 6
ifIndex.7 = 7
	ifDescr.1 = MgmtEth0/RP1/CPU0/0
	ifType.1 = ethernetCsmacd(6)
	ifAdminStatus.1 = up(1)
	ifDescr.2 = Null0
	ifType.2 = other(1)
	ifAdminStatus.2 = up(1)
	ifDescr.3 = SONET0/2/0/0
	ifType.3 = sonet(39)
	ifAdminStatus.3 = up(1)
	ifDescr.4 = POS0/2/0/0
	ifType.4 = pos(171)
	ifAdminStatus.4 = down(2)

Sparse Tables

A *sparse table* is a table where only a subset of columnar objects is instantiated for each row. In MIB tables this occurs when certain objects are not instantiated for certain rows, because it is either not applicable or not available at that time (for example, in the ifTable, some of the counters are not applicable for tunnel interfaces). Sparse MIB tables are potential sources of low performance during table traversal, especially with row-wise access.

Table E-2 is a snapshot from the ifTable of IF-MIB showing a sparse entry for SONET0/2/0/0.

Table E-2 Example of sparse entry from IF-MIB

ifDescr	isInOctets
MgmtEth0/RP1/CPU0/0	5036844
SONET0/2/0/0	
POS0/2/0/0	1024

During row-wise traversal each request would contain objects that are of interest to the OSS. MIB implementation processes each of the requests one by one. For each of these objects it needs to identify the ‘next’ instance and the value for this instance. Identifying “next” instance could be a CPU intensive operation as it might involve sorting or searching internal data structures or even external data structures in distributed environments. As an optimization, this “next” instance identification needs to be done only once per request containing multiple objects from the same table. This optimization would break if one of the objects is not applicable or not available for already identified “next” instance. This will result in further searching until the system identifies a new “next” instance for that object.

Note The best approach is to avoid these sparse objects during row-wise traversal of MIB tables by avoiding GET-NEXT over unsupported objects.

A row-wise GET on some of the selected objects from IF-MIB (ifTable) is shown below. Each request contains two objects. Third response onwards contains a “jump” over sparse object and this “jump” requires additional searching every time.
bash-2.03$ getmany -v2c 12.25.26.10 public ifDescr ifInOctets

```
ifDescr.1 = MgmtEth0/RP1/CPU0/0
ifInOctets.1 = 5072278
ifDescr.2 = Null0
ifInOctets.2 = 0
ifDescr.3 = SONET0/2/0/0
ifInOctets.4 = 0
ifDescr.4 = POS0/2/0/0
ifInOctets.6 = 0
ifDescr.5 = SONET0/2/0/1
ifInOctets.8 = 0
```

Requests Addressed to Interleaved Objects

Accessing all the required objects of a row in a single request is much more efficient than making multiple queries for each. Single requests can contain objects for multiple rows also. It is important to note that the request should contain objects with the same instance in sequence. Interleaving objects with different instances would result in issues similar to sparse tables and multiple retrieval of same information from feature related process.

The example below shows an SNMP-GET request performed on IF-MIB (ifTable). The request contains objects from two different interfaces. Object instances are interleaved in the request.

```
bash-2.03$ getone -v2c 12.25.26.10 public ifDescr.1 ifType.2 ifType.1 ifDescr.2
ifDescr.1 = MgmtEth0/RP1/CPU0/0
ifType.2 = other(1)
ifType.1 = ethernetCsmacd(6)
ifDescr.2 = Null0
bash-2.03$
```

Rearranging this request with consecutive objects for the same instance would be more efficient in terms of processing.

```
bash-2.03$ getone -v2c 12.25.26.10 public ifDescr.1 ifType.1 ifType.2 ifDescr.2
ifDescr.1 = MgmtEth0/RP1/CPU0/0
ifType.1 = ethernetCsmacd(6)
ifType.2 = other(1)
ifDescr.2 = Null0
bash-2.03$
```

Large Tables

Any MIB table that is expected to contain 100 entries is considered a large table. IF-MIB, IPFORWARD-MIB, and IP-MIB have examples of large tables. Traversing these tables should be performed in a more intelligent way as it can consume a lot of resources. OSSs should make use of any existing additional objects which provide overall information about the table, for example, an object describing number of entries or last modified time. Last modified time could be used to identify whether there have been any changes to the table since last retrieval. An object representing number of entries could be used to split the retrieval into multiple sets of smaller discoveries separated by time. For example:

- IP-FORWARD-MIB::ipCidrRouteNumber
- IP-MIB::ipv4InterfaceTableLastChange
- ENTITY-MIB::entLastChangeTime

Static Data

Some tables actually contain data which do not change during a management session. Data which are almost static should be retrieved at the beginning once and then no longer accessed. In other words an SNMP agent should not be used as a repository to be polled for constant data. Of course a smart OSS should know when it is time to fetch the data again, if some event takes place of OIR. If these data are fetched once only, this saves CPU time for other requests as well as useless DCN load.

• ENTITY-MIB::entPhysicalTable

Use of SNMP views

Access to MIB tables or objects can be optimized with the use of appropriate SNMP view configurations. This is useful when an OSS makes frequent retrieval of very large tables and there is no direct control on it. To achieve this, an SNMP view has to be configured by excluding respective MIB sub-trees and attaching this view to the SNMP community name or group name being used. This approach can also be applied in cases where certain MIB implementations being polled have to be prevented (for example, very slow response or impacting the performance of other MIBs). The following shows a view configuration and associated community configuration which prevents IP-FORWARDMIB (ipCidrRouteTable) and IP-MIB (ipNetToMediaTable) being polled. SNMP query made with community string “mycom” will not access these two tables.

- snmp-server view cutdown 1.3.6.1.2.1.4.22 excluded
- snmp-server view cutdown 1.3.6.1.2.1.4.24.4 excluded
- snmp-server community mycom view cutdown

Cut down views can also be created by excluding everything and including only what is required by OSSs.

Table Access Recommendations

1. In a sparse table do specific GETs and avoid using GET-NEXT over unsupported objects.
2. In non-sparse tables, use GET or GET-BULK to acquire all related objects in one request instead of using GET-NEXT to walk a table.
3. Consider if objects are interleaved and use this information to efficiently make GET requests.
4. When possible, use objects which provide information about large tables instead of accessing the whole table.
5. Do not retrieve nearly static data more often than absolutely necessary (for example, at the start of an OSS session).
6. Use SNMP views to carefully select what information is retrieved.

Multiple OSS

Parallel access of the same MIB from multiple polling stations can lead to slower response and higher CPU utilization. This is evident especially in the case of large tables, where multiple stations access different part of the same MIB table. In distributed environments most of the MIB implementations would have some kind of look-ahead caching done. This holds well if requests fall in this cache. This would lead to performance issues because of repeated cache misses OTHERs.

Multiple OSS recommendation: Avoid having multiple OSSs polling the same MIB—instead share the data between the OSSs.

MIB Specific Functionality

Some MIBs provide key features that permit improved NMS/OSS operation.

One such feature is IF-MIB ifindex persistence feature (also relevant for other MIBs that use the ifIndex such as Etherlike-MIB). This feature permits ifindices to remain constant across reboots, which prevents the need for the NMS/OSS to perform interface rediscovery after a reboot. Another similar feature is the ENTITY-MIB entphyscialindex persistence (also relevant for MIBs that extend the Entity-MIB such as the CISCO-ENTITY-FRUCONTROL-MIB). Similar to ifindex persistence, this feature permits entphysical indices to persist across reboots. Finally, the CISCO-CLASS-BASED-QOS-MIB also has an index persistence function, which prevents the need to rediscover QoS indices after reboot. To obtain information on the configuration of these features see the IOS-XR System Management Command Reference documents at:

http://www.cisco.com/en/US/products/ps5845/prod_command_reference_list.html

General Performance Considerations and Tunable Parameters

Performance tuning and considerations for SNMP stem from two different perspectives:

- Desire to adjust/improve the performance of SNMP in specific operations
- Desire to limit impact of SNMP performance on overall device operations

IOS-XR provides a number of options to assist in both of these areas. IOS-XR supports both in band and out of band SNMP operation. In band processing can be controlled via LPTS (control plane policing) in the same way as other control plane operations. Both in band and out of band operation can further be tuned by the following mechanisms:

- Request throttling [snmp-server throttle-time]
- Queue length throttling (trap only) [snmp-server queue-length]

Note This only affects the outgoing trap queue, not the incoming request queue.

- SNMP Overload Control [snmp-server overload]
- Protection of critical operations (such as routing convergence) from SNMP CPU overuse.

To obtain information on the configuration of these features see the IOS-XR System Management Command Reference documents at:

http://www.cisco.com/en/US/products/ps5845/prod_command_reference_list.html

Note that many SNMP performance considerations are related to specific MIB implementations, but the above system wide settings do have some affect on performance.

MIB Specific Performance Considerations and Tunable Parameters

MIB polling performance is frequently associated with the implementation of a specific MIB and there are some tunable parameters for specific MIBs. For the IF-MIB, caching support is configured using the **snmp-server ifmib stats cache** command.

Note This command only affects the SNMP specific interface statistic operation, the behavior of the IOS-XR internal statistics collection mechanism in IOS-XR is not controllable from SNMP.

The CISCO-CLASS-BASED-QOS-MIB also has caching support, this is configured via the **snmp-server mibs cbqosmib cache** command. The obvious tradeoff in the use of these caches is faster performance versus most recent data. To obtain information on the configuration of these features see the IOS-XR System Management Command Reference documents at:

http://www.cisco.com/en/US/products/ps5845/prod_command_reference_list.html