Performance Monitoring using SNMP for Nokia Platforms

IPSO v3.4.1 White Paper

Part No. N450681001, Rev A

Published May 31, 2002

Contents

Tables

1.0 Objective

This white paper is the first of a series of documents describing the Simple Network Management Protocol (SNMP) management information base (MIB) objects necessary to gather performance data for Nokia Internet Communications (Nokia) appliances.

Note
The Nokia appliances this document applies to are the IP100, IP300, IP400, IP500, IP600, and IP700 series platforms.

The goal of this document is to identify MIB objects that help you collect useful performance data from your Nokia appliances. Once you collect performance data, you can analyze the information to help identify traffic patterns on your network and plan for future network capacity. This document addresses *what* information you should collect rather than *how* to collect and analyze it.

This document contains several tables that describe MIB objects that help you monitor network performance. The tables also list the object identifier (OID), which is the address of the object in dotted-decimal format, and a sample return value for the object. For a list of all the tables in this document, see "Tables" on page v.

The objects described in this document can help you monitor the performance of your network. Although some of the objects are also useful for fault monitoring and network characterization, this document concentrates on using these objects for performance monitoring. Fault monitoring and network characterization are discussed in separate documents.

This document assumes you are familiar with:

- Abstract Syntax Notation One (ASN.1)
- SNMP
- At least one SNMP-based network management tool
- The MIB tree

Note
Nokia appliances provide secure communications from management stations to the appliance, through the command line (via SSH versions 1 and 2), a web browser (via SSL and TLS), and SNMP (SNMP v3)

All MIB objects in this document are available in IPSO 3.4.1-FCS11. Earlier and later versions of IPSO may not contain all objects discussed here.

For a list of RFCs that define the objects in this document, see Section 6.0, "References" on page 43.

2.0 Overview

The SNMP protocol is the de facto standard for network fault monitoring, which allows the network administrator to monitor the health of the network at any moment. SNMP also plays a key role in performance monitoring, which helps the network administrator understand how the network is performing as a whole. Both of these SNMP activities are important in determining the current state and future capabilities of the network.

Network performance data helps you understand the future capacity of the network to support the increasing demands placed upon it. One of the reasons it is difficult to monitor a network is that a network comprises many components, each with different operating characteristics and monitoring needs. Some network elements and properties you might monitor are:

- WAN and LAN interfaces on routers
- Router operating state (memory, temperature)
- Switch (or hub) port traffic level
- Rack-mounted modem usage
- Remote access server activity (Radius, SecurID, TACACS, and so on)
- DNS activity
- Mail queues

- Host components (for example, disk, load average, swap space)
- Application components

Some of these components (for example, DNS, hosts) are not directly related to the health of the physical network but provide other important information. Additionally, networks often contain different administrative groups (for example, the *host group*, the *network group*, and the *security group*) that might or might not work together closely, which makes performance monitoring even more complex.

The three most common methods of approaching both fault and performance monitoring are:

- None
- Local agents
- SNMP

Unfortunately, the most popular method is not to monitor the network at all. It is the easiest method to manage and install. If a problem occurs on the network, the end users become the method of notification. This is not a recommended manner of maintaining a network. The correct approaches either use proprietary local agents on each network element or use SNMP.

2.1 Local Agents

Local agents typically use the native utilities of a network element to gather information about the hardware and software, then forward the information to a data collector. Local agent solutions do not scale well to medium and large installations. As the number of network elements grows, the administrative effort necessary to maintain the configuration becomes increasingly difficult. Additionally, the protocols used to communicate between the local agents and the central data collector are not standardized, nor are they typically secured. Data presentation and storage is often inconsistent, and any local agent monitoring scheme will have difficulty with new hardware in any heterogeneous environment.

2.2 SNMP

Simple Network Management Protocol (SNMP) is a set of standards that most network equipment supports. A complete implementation of SNMP provides all the information necessary to characterize the fault status and performance of any network element. The aggregated network information characterizes the network as a whole. Moreover, the network information is represented in a hierarchical fashion. This hierarchical representation allows the information base to be expanded without adversely affecting previous installed versions. This is accomplished by allowing additional branches to be added without changing the existing information base.

Any management objects that are not present on all network elements are not described in the standard management information base (MIB); the equipment manufacturer should provide it in a MIB file. These enterprise MIBs complement standardized MIBs. Since SNMP uses a well-known port (port 161) for all communications, and all communications are UDP based, any host with the proper software is capable of sending and receiving SNMP based information. This capability also allows you to craft the network security policy so that all SNMP traffic is routed correctly and efficiently.

The preferred method for monitoring any homogeneous or heterogeneous network is SNMP. The SNMPv3 standard lets you secure all communication between a network element and its data collector.

3.0 Network Characterization

In order to monitor performance, you must first obtain some basic information about your system. This section discusses the MIB objects that help you collect information about the status and setup of your network.

Section 4.0, “Performance Monitoring” on page 16, describes the MIB objects that allow you to collect performance data from your Nokia appliance. Once you know the behavior of a network element, you can collect specific data from that element to identify performance trends.

3.1 System Identity

The objects in this section are defined in RFC 1907.

Although system identification values do not provide information about performance or fault, they are essential in confirming that the values being retrieved are from the correct host. SNMP extracts all configuration information from the target appliance. Table 1 contains the list of objects (name, OID, sample value, description) that provide the system information. For Nokia appliances, the system description (`sysDescr`) variable is generated from the host information (for example, `uname -a`); the system name (`sysName`) value is derived from the host name. These objects cannot be set using SNMP. You must set them correctly for each appliance using Voyager or the values the SNMP objects return will be incorrect.

Table 1 System Information Objects

Object name	OID	Sample value	Description
`sysDescr`	1.3.6.1.2.1.1.1.0	IP740 rev 00, IPSO serine 3.4.1-FCS5 releng 826 08.18.2001-053000 i386	System description
`sysUptime`	1.3.6.1.2.1.1.3.0	53 days, 00:27:24.73	System uptime
`sysContact`	1.3.6.1.2.1.1.4.0	Joe Admin [Joe.Admin@company.dom]	System contact
`sysName`	1.3.6.1.2.1.1.5.0	serine-gw	Host name
`sysLocation`	1.3.6.1.2.1.1.6.0	serine-gw.company.dom (service integration point)	System location
`sysServices`	1.3.6.1.2.1.1.7.0	76	Indicates the set of services that the appliance potentially offers.

The `sysServices` object indicates the set of services that the appliance potentially offers. This value is the sum of a series of numbers. The value of this object starts at zero. Then, for every layer, **L**, in the range 1 through 7,

that this appliance performs transactions for, add 2 raised to (**L** - 1) to the sum. For layer descriptions, see Table 2.

For example, a node that performs only routing (layer 3) functions has a value of 4:

$$2^{(3-1)} = 4$$

In contrast, a host node that offers application services (layer 7) has a value of 72:

$$2^{(4-1)} + 2^{(7-1)} = 72$$

Note
Application services (layer 7) use TCP (layer 4), so you must include both layers in the equation.

Table 2 Internet Suite Protocol Layer Overview

Layer	Functionality
1	Physical (for example, repeaters)
2	Datalink/subnetwork (for example, bridges)
3	Internet (for example, supports IP)
4	End-to-end (for example, supports TCP)
7	Applications (for example, supports SMTP)

For systems that include OSI protocols, you can also count layers 5 and 6.

3.2 Network Interface Properties

The objects in this section are defined in RFC 2233.

To observe performance, identify the number of interfaces and their indices or identification number. The `ifNumber` object shows the number of network

interfaces (regardless of their current state) present on this system. Each network interface is assigned a unique value, greater than zero. The `ifIndex.?` objects store these values.

Note
When an object name is followed by a ".?" notation, this indicates that the object is variable depending upon the host configuration. In other words, the object instantiation depends on the host configuration.

Table 3 Interface and Index Objects

Object name	OID	Sample value	Description
`ifNumber`	1.3.6.1.2.1.2.1.0	7	Number of network interfaces present on a system
`ifIndex.?`	1.3.6.1.2.1.2.2.1.1.?	1	Value of the network interface

Once you identify the interface indices, record the properties for each interface:

- Description
- Speed
- MTU size
- Physical address
- Status (administrative and operational)
- Time since the interface settings were last changed

The `ifDescr.?` object describes the network interface as a string that contains information about the interface. This string includes the name of the manufacturer, the product name, and the version of the interface hardware and software.

The `ifType.?` object identifies the interface type. There are 108 possible physical and logical interface values for the `ifType` object.

The `ifSpeed.?` object contains an estimate of the current data rate of the interface in bits per second. For interfaces that do not vary in data rate, or for those where no accurate estimation can be made, this object contains the nominal data rate. For a sublayer that has no concept of data rate, this object is zero.

The `ifMTU.?` object contains the size of the largest packet that can be sent, or received, on the interface, specified in octets. For interfaces that are used for transmitting network datagrams, this is the size of the largest network datagram that can be sent on the interface.

The `ifPhysAddress.?` object stores the address of the interface at its protocol sublayer. For example, for an 802.x interface, this object normally contains a media access control (MAC) address. The media-specific MIB of the interface defines the bit and byte ordering and the format of the value of this object. For interfaces that do not have such an address (for instance, a serial line), this object contains an octet string of zero length.

The `ifAdminStatus` indicates whether that interface has been enabled or disabled at the operating system level. This object indicates the explicit management action with respect to the interface status. This object is independent of the network connection of the interface. The testing(3) state indicates that no operational packets can be passed. When a managed system initializes, all interfaces start with `ifAdminStatus` in the down(2) state. As a result of either explicit management action, or because of configuration information retained by the managed system, the `ifAdminStatus` changes to either the up(1) or testing(3) states (or remains in the down(2) state).

The `ifOperStatus.?` object provides the current operational state of the interface. It indicates the network status with respect to the interface. The value of this object is a combination of the administrative status and network connectivity. The testing(3) state indicates that no operational packets can be passed. If `ifAdminStatus` is down(2) then `ifOperStatus` is down(2). If `ifAdminStatus` is changed to up(1) then `ifOperStatus` changes to up(1). If the interface is ready to transmit and receive network traffic, it changes to dormant(5). If the interface is waiting for external actions (such as a serial line waiting for an incoming connection), it remains in the down(2) state if and

only if a fault prevents it from going to the up(1) state. The interface remains in the notPresent(6) state if components (typically, hardware) are missing.

The `ifLastChange.?` object contains the value of `sysUpTime` at the time of the last creation or deletion of an entry in the `ifTable`. If the number of entries is unchanged since the last reinitialization of the local network management subsystem, this object contains a zero value.

Table 4 Interface Property Objects

Object name	OID	Sample value	Description
`ifDescr.?`	1.3.6.1.2.1.2.2.1.2.?	eth-s1p1c0	Describes the network interface as a string that contains information about the interface.
`ifType.?`	1.3.6.1.2.1.2.2.1.3.?	ethernetCsmacd(6)	Describes the type of interface.
`ifSpeed.?`	1.3.6.1.2.1.2.2.1.5.?	100000000	Estimate of the current bandwidth of the interface in bits per second.
`ifMTU.?`	1.3.6.1.2.1.2.2.1.4.?	1500	Size of the largest packet that can be sent, or received, on the interface, specified in octets.
`ifPhysAddress.?`	1.3.6.1.2.1.2.2.1.6.?	0x00a08e20388c	Address of the interface at its protocol sublayer.
`ifAdminStatus.?`	1.3.6.1.2.1.2.2.1.7.?	up	State of the network interface.
`ifOperStatus.?`	1.3.6.1.2.1.2.2.1.8.?	up	Current operational state of the interface.
`ifLastChange.?`	1.3.6.1.2.1.2.2.1.9.?	10.84 seconds	Value of `sysUpTime` at the time of the last creation or deletion of an entry in the `ifTable`.

4.0 Performance Monitoring

You can determine the performance of any Nokia appliance by gathering select SNMP values from the appliance. You can group the performance values identified in the following table in many ways to make them meaningful, depending on the network. The most significant method is to segregate the interface-specific performance values (throughput, discards, and so on) and the traffic-type performance values.

4.1 Network Interface Values

The objects in this section are defined in RFC 2233.

Monitor any parameter that affects the overall throughput of network traffic, as well as the volume, direction, and type of network activity experienced for each network interface. Include the total traffic passed through each appliance interface separated in the inbound and outbound directions. A Nokia appliance provides statistical information for all network interfaces installed on the appliance.

You can extract the total number of octets (eight-bit bytes) received on an interface, including framing characters, from the appliance. You can also determine the inbound and outbound traffic volume for each interface on the network appliance.

Table 5 Octet Count Objects

Object name	OID	Sample value	Description
`ifInOctets.?`	1.3.6.1.2.1.2.2.1.10.?	33775574	Number of octets received on an interface.
`ifOutOctets.?`	1.3.6.1.2.1.2.2.1.16.?	40015479	Number of octets sent on an interface.

Additionally, collect all discards and errors at each interface for the inbound and outbound traffic.

The `ifInDiscards.?` and `ifOutDiscards.?` objects show the number of inbound, or outbound, packets that are chosen to be discarded even though no errors were detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet is to make buffer space available.

For packet-oriented, character-oriented, or fixed-length interfaces, the `ifInErrors.?` and `ifOutErrors.?` objects contain the number of inbound or outbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.

Table 6 Packet Discard and Error Objects

Object name	OID	Sample value	Description
`ifInDiscards.?`	1.3.6.1.2.1.2.2.1.13.?	0	Number of inbound (no error) packets chosen to be discarded.
`ifOutDiscard.?`	1.3.6.1.2.1.2.2.1.19.?	0	Number of outbound (no error) packets chosen to be discarded.
`ifInErrors.?`	1.3.6.1.2.1.2.2.1.14.?	0	Number of inbound packets that contained errors and were discarded.
`ifOutErrors.?`	1.3.6.1.2.1.2.2.1.20.?	114	Number of outbound packets that contained errors and were discarded.

A few other interface-specific performance values can be gathered to allow for performance evaluation.

The `ifInNUcastPkts.?` and `ifOutNUcastPkts.?` objects describe the number of packets that a sublayer delivers to a higher (sub)layer, which were addressed to a multicast or broadcast address at this sublayer. The `ifInUcastPkts.?` and `ifOutUcastPkts.?` objects describe the number of network packets that a sublayer delivers to a higher (sub)layer, which were *not* addressed to a multicast or broadcast address at this sublayer.

The object `ifInUnknownProtos.?` contains the number of packets received by an interface and discarded because of an unknown or unsupported

protocol. This process applies to packet-oriented interfaces, and for character-oriented or fixed-length interfaces that support protocol multiplexing. For any interface that does not support protocol multiplexing, this counter is always zero.

The `ifOutQLen.?` object stores the length (in number of packets) of the output packet queue.

Table 7 Additional Packet Objects

Object name	OID	Sample value	Description
`ifInUcastPkts.?`	1.3.6.1.2.1.2.2.1.11.?	386159	Number of packets that a sublayer delivers to a higher (sub)layer, which were addressed to a multicast or broadcast address at the sublayer.
`ifOutUcastPkts.?`	1.3.6.1.2.1.2.2.1.17.?	366211	Number of packets that a sublayer delivers to a higher (sub)layer, which were addressed to a multicast or broadcast address at the sublayer.
`ifInNUcastPkts.?`	1.3.6.1.2.1.2.2.1.12.?	15910	Number of network packets that a sublayer delivers to a higher (sub)layer, which were *not* addressed to a multicast or broadcast address at this sublayer
`ifOutNUcastPkts.?`	1.3.6.1.2.1.2.2.1.18.?	0	Number of network packets that a sublayer delivers to a higher (sub)layer, which were *not* addressed to a multicast or broadcast address at this sublayer.

Table 7 Additional Packet Objects

Object name	OID	Sample value	Description
`ifInUnknownProtos.?`	1.3.6.1.2.1.2.2.1.15.?	0	Number of packets received and discarded because of an unknown or unsupported protocol.
`ifOutQLen.?`	1.3.6.1.2.1.2.2.1.21.?	0	Length of the output packet queue.

These SNMP values provide a subnet-specific indication of the network performance of the appliance. To understand the true performance implications over a well-defined time period, collect and analyze these values.

4.2 Network Traffic by Protocol Type

Network traffic contains of a variety of protocols (for example, IP, ICMP, TCP, and UDP). To get an overall view of your network performance, you must examine the performance of each protocol that your network uses. For each protocol, you use a different set of MIB objects. This section discusses the MIB objects by protocol type that can help you monitor network traffic performance.

4.2.1 IP Values

The objects in this section are defined in RFC 2011 (IP) and RFC 2096 (IP Forwarding).

You can collect IP performance values from the objects listed in this section. The `ipInReceives` counts the total number of input datagrams received from interfaces, including those received in error. The `ipInDelivers` object counts the total number of input datagrams successfully delivered to IP user protocols, including ICMP.

The ipOutRequests object counts the total number of IP datagrams that local IP user protocols, including ICMP, supplied to IP in requests for transmission. This counter does not include any datagrams counted in ipForwDatagrams. The ipOutDiscards object shows the number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but that were discarded. This counter includes datagrams counted in ipForwDatagrams (if any such packets met this discretionary discard criterion) objects. These values provide an overall measure of the network traffic to and through the appliance.

Table 8 IP Performance Value Objects

Object name	OID	Sample value	Description
ipInReceives	1.3.6.1.2.1.4.3.0	1518788	Number of input datagrams received from interfaces, including those received in error.
ipInDelivers	1.3.6.1.2.1.4.9.0	52964	Number of input datagrams successfully delivered to IP user protocols, including ICMP.
ipInDiscards	1.3.6.1.2.1.4.8.0	0	Number of input IP datagrams for which no problem was encountered to prevent their transmission to their destination, but that were discarded.
ipOutRequests	1.3.6.1.2.1.4.10.0	58751	Total number of IP datagrams that local IP user protocols, including ICMP, supplied to IP in requests for transmission.
ipOutDiscards	1.3.6.1.2.1.4.11.0	0	Number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but that were discarded.

The SNMP objects described in the following section can collect additional IP information.

The ipInHdrErrors object contains the number of input datagrams discarded because of errors in their IP headers, including:

- Bad checksums
- Version number mismatch
- Other format errors
- Time-to-live exceeded
- Errors discovered in processing their IP options

The ipInAddrErrors object contains the number of input datagrams discarded because the IP address in the IP header destination field was not a valid address to be received. This count includes invalid addresses (for instance, 0.0.0.0) and addresses of unsupported classes (Class E). For entities that are not IP Gateways and therefore do not forward datagrams, this counter includes datagrams discarded because the destination address is not a local address.

The ipInUnknownProtos object contains the number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol.

The ipOutNoRoutes object counts the number of IP datagrams discarded because no route could be found to transmit them to their destination. This counter includes any packets counted in ipForwDatagrams that meet this *no-route* criterion; this includes any datagrams that a host cannot route because *all* of its default gateways are down.

The ipForwarding object contains the indication of whether an entity is acting as an IP gateway with respect to the forwarding of datagrams received by, but not addressed to, the entity. IP gateways forward datagrams; IP hosts do not (except those source-routed through the host). For some managed nodes, this object can take on only a subset of the values possible. Accordingly, it is appropriate for a local agent to return a badValue response if a management station attempts to change this object to an inappropriate value.

The ipForwDatagrams object contains the number of input datagrams for which this network entity was not their final IP destination, as a result of

which an attempt was made to find a route to forward them to that final destination. For network appliances that do not act as IP gateways, this counter includes only those packets that were source routed through this entity, and the source route option processing was successful.

The `ipRoutingDiscards` object counts the number of routing entries that were chosen to be discarded, even though they are valid. One possible reason for discarding such an entry is to free buffer space for other routing entries.

Table 9 Additional IP Objects

Object name	OID	Sample value	Description
`ipInHdrErrors`	1.3.6.1.2.1.4.4.0	48	Number of input datagrams discarded because of errors in their IP headers.
`ipInAddrErrors`	1.3.6.1.2.1.4.5.0	0	Number of input datagrams discarded because the IP address in their IP header destination field was not a valid address to be received.
`ipInUnknownProtos`	1.3.6.1.2.1.4.7.0	0	Number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol.
`ipOutNoRoutes`	1.3.6.1.2.1.4.12.0	0	Number of IP datagrams discarded because no route could be found to transmit them to their destination.
`ipForwarding`	1.3.6.1.2.1.4.1.0	On (1)	Indicates whether an entity is acting as an IP gateway with respect to the forwarding of datagrams received by, but not addressed to, the entity.

Table 9 Additional IP Objects

Object name	OID	Sample value	Description
`ipForwDatagrams`	1.3.6.1.2.1.4.6.0	250827493	Number of input datagrams for which this network entity is not their final IP destination.
`ipRoutingDiscards`	1.3.6.1.2.1.4.23.0	0	Number of valid routing entries that were chosen to be discarded.

The next set of objects characterize the IP traffic at the fragmentation and reassembly level.

The `ipReasmReqds` object stores the number of IP fragments received that needed to be reassembled for this network appliance.

The `ipReasmOks` object contains the number of IP datagrams successfully reassembled. The `ipReasmFails` object contains the number of failures detected by the IP reassembly algorithm (for example, timed out, errors). However, this is not necessarily a count of discarded IP fragments since some algorithms (notably the algorithm in RFC 815) can lose track of the number of fragments by combining them as they are received.

Table 10 Reassembly Level IP Objects

Object name	OID	Sample value	Description
`ipReasmReqds`	1.3.6.1.2.1.4.14.0	0	Number of IP fragments received that needed to be reassembled for this network appliance.
`ipReasmOks`	1.3.6.1.2.1.4.15.0	0	Number of IP datagrams successfully reassembled.
`ipReasmFails`	1.3.6.1.2.1.4.16.0	0	Number of failures that the IP reassembly algorithm detects.

The ipFragOks object contains the number of IP datagrams that were successfully fragmented at this network appliance. The ipFragFails object stores the number of IP datagrams that were discarded because they needed to be fragmented at this entity but could not be fragmented (for example, because their *Don't Fragment* flag was set). The ipFragCreates object contains the number of IP datagram fragments that were generated as a result of fragmentation at this network element.

Table 11 Fragmentation Level IP Objects

Object name	OID	Sample value	Description
ipFragOks	1.3.6.1.2.1.4.17.0	0	Number of IP datagrams successfully fragmented on the network element.
ipFragFails	1.3.6.1.2.1.4.18.0	0	Number of IP datagrams discarded because they needed to be (but were not) fragmented on the network element.
ipFragCreates	1.3.6.1.2.1.4.19.0	0	Number of IP datagram fragments generated as a result of fragmentation at the network element.

The objects described thus far allow you to characterize all IP network traffic. Some of these objects provide extraneous and little-needed data. The complete set of objects are provided for future reference.

4.2.2 ICMP Values

The objects in this section are defined in RFC 1213.

Inbound and outbound ICMP traffic on an appliance provides routing environment information.The icmpInMesgs object contains the total number of ICMP messages a network element has received. This counter also includes all ICMP messages that icmpInErrors counts. The icmpInErrors object contains the number of ICMP messages that a network element received but determined to have ICMP specific errors (for example, bad ICMP checksums, bad length).

The `icmpOutMsgs` object counts the total number of ICMP messages a network element attempted to send. This counter includes all ICMP messages that `icmpOutErrors` counts. The `icmpOutErrors` object counts the number of ICMP messages a network element did not send because of problems discovered within ICMP, such as a lack of buffers. This value should not include errors discovered outside the ICMP layer, such as the inability of IP to route the resulting datagram. Some implementations might have no errors that contribute to the `icmpOutMsgs` counter value.

Table 12 ICMP Message Objects

Object name	OID	Sample value	Description
`icmpInMesgs`	1.3.6.1.2.1.5.1.0	4897	Number of ICMP messages a network element has received.
`icmpInErrors`	1.3.6.1.2.1.5.2.0	1	Number of ICMP messages that a network element received but determined as having ICMP specific errors.
`icmpOutMsgs`	1.3.6.1.2.1.5.14.0	7242	Number of ICMP messages a network element attempted to send.
`icmpOutErrors`	1.3.6.1.2.1.5.15.0	0	Number of ICMP messages a network element did not send because of problems discovered within ICMP.

Several optional ICMP performance values can provide additional granularity to the performance evaluation of an appliance. Table 13 contains inbound

ICMP performance objects, and Table 14 shows outbound ICMP performance objects.

Table 13 Inbound ICMP Performance Objects

Object name	OID	Sample value	Description
icmpInDestUnreachs	1.3.6.1.2.1.5.3.0	25	Number of ICMP Destination Unreachable messages received.
icmpInTimeExcds	1.3.6.1.2.1.5.4.0	0	Number of ICMP Time Exceeded messages received.
icmpInParmProbs	1.3.6.1.2.1.5.5.0	0	Number of ICMP Parameter Problem messages received.
icmpInSrcQuenchs	1.3.6.1.2.1.5.6.0	0	Number of ICMP Source Quench messages received.
icmpInRedirects	1.3.6.1.2.1.5.7.0	4	Number of ICMP Redirect messages received.
icmpInEchos	1.3.6.1.2.1.5.8.0	302977	Number of ICMP Echo (request) messages received.
icmpInEchoReps	1.3.6.1.2.1.5.9.0	0	Number of ICMP Echo Reply messages received.

.

Table 14 Outbound ICMP Performance Objects

Object name	OID	Sample value	Description
icmpOutDestUnreachs	1.3.6.1.2.1.5.16.0	318293	Number of ICMP Destination Unreachable messages sent.
icmpOutTimeExcds	1.3.6.1.2.1.5.17.0	48	Number of ICMP Time Exceeded messages sent.

Table 14 Outbound ICMP Performance Objects (*continued*)

Object name	OID	Sample value	Description
icmpOutParmProbs	1.3.6.1.2.1.5.18.0	0	Number of ICMP Parameter Problem messages sent.
icmpOutSrcQuenchs	1.3.6.1.2.1.5.19.0	0	Number of ICMP Source Quench messages sent.
icmpOutRedirects	1.3.6.1.2.1.5.20.0	0	Number of ICMP Redirect messages sent.
icmpOutEchos	1.3.6.1.2.1.5.21.0	0	Number of ICMP Echo (request) messages sent.
icmpOutEchoReps	1.3.6.1.2.1.5.22.0	302977	Number of ICMP Echo Reply messages sent.

Most of these ICMP counters are useful as secondary information.

Note
It is easier to troubleshoot by monitoring the interface that is sending out the ICMP packets, than to try to watch all systems for inbound problems.

4.2.3 UDP Values

The objects in this section are defined in RFC 2013.

The UDP values relevant to performance monitoring are the inbound and outbound packets and UDP related errors.

Table 15 UDP Objects

Object name	OID	Sample value	Description
udpInDatagrams	1.3.6.1.2.1.7.1.0	41151	Number of UDP datagrams delivered to UDP users.
udpOutDatagrams	1.3.6.1.2.1.7.4.0	40339	Number of UDP datagrams sent from this network element.
udpNoPorts	1.3.6.1.2.1.7.2.0	0	Number of received UDP datagrams that had no application at the destination port.
udpInErrors	1.3.6.1.2.1.7.3.0	0	Number of received UDP datagrams that could not be delivered (for reasons other than the lack of an application at the destination port).

Performance monitoring is less of a concern for UDP than for TCP traffic. Any application that uses UDP handles the errors and retransmissions. However, these few UDP objects must be monitored as part of the overall performance profile.

4.2.4 TCP Values

The objects in this section are defined in RFC 2012.

You can summarize the TCP values from the active opens, passive opens, and the current established connections values. You can also evaluate the inbound and outbound segments.

Table 16 TCP Connection Objects

Object name	OID	Sample value	Description
tcpActiveOpens	1.3.6.1.2.1.6.5.0	25	Number of times TCP connections made a direct transition to the SYN-SENT state from the CLOSED state.
tcpPassiveOpens	1.3.6.1.2.1.6.6.0	333	Number of times TCP connections made a direct transition to the SYN-RCVD state from the LISTEN state.
tcpEstabResets	1.3.6.1.2.1.6.8.0	5	Number of times TCP connections made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.
tcpCurrEstab	1.3.6.1.2.1.6.9.0	0	Number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.

Table 17 TCP Inbound and Outbound Segment Objects

Object name	OID	Sample value	Description
tcpInSegs	1.3.6.1.2.1.6.10.0	8107	Number of segments received, including those received in error and segments received on currently established connections.
tcpInErrs	1.3.6.1.2.1.6.14.0	0	Number of segments received in error (for example, bad TCP checksums).

Table 17 TCP Inbound and Outbound Segment Objects

Object name	OID	Sample value	Description
tcpOutSegs	1.3.6.1.2.1.6.11.0	10054	Number of segments sent, including those on current connections but excluding those containing only retransmitted octets.
tcpOutRsts	1.3.6.1.2.1.6.15.0	0	Number of TCP segments sent that contain the RST flag.

The general TCP errors that have no sense of traffic direction can provide valuable data about the appliance performance.

The tcpAttemptFails object describes the number of times TCP connections made a direct transition to the *CLOSED* state from either the *SYN-SENT* state or the *SYN-RCVD* state, plus the number of times TCP connections made a direct transition to the *LISTEN* state from the *SYN-RCVD* state. The tcpRetransSegs object represents the total number of segments retransmitted (for example, the number of TCP segments transmitted that contain one or more previously transmitted octets).

Table 18 General TCP Error Objects

Object name	OID	Sample value	Description
tcpAttemptFails	1.3.6.1.2.1.6.7.0	169144	Number of times TCP connections made a direct transition to various states.
tcpRetransSegs	1.3.6.1.2.1.6.12.0	696	Number of segments retransmitted.

The TCP objects in Table 18 allow you to characterize all TCP connections. Some of these objects provide extraneous and little-needed data. The complete set of objects are provided for future reference.

4.2.5 Frame Relay

The objects in this section are defined in RFC 2115.

The last network interface type to consider in performance monitoring is any frame relay interface on the appliance. The basic performance values for frame relay are the number of sent and received frames or octets.

The information in this section enables you to fully characterize all frame relay virtual circuits. Some of these objects provide extraneous and little-needed data. The complete set of objects are provided for future reference.

You can determine frame relay physical interface characteristics from the eleven objects in Table 19.

Table 19 Frame Relay Object Descriptions

Object Name	Description
`frDlcmiIfIndex`	Indicates the `ifIndex` value of the corresponding `ifEntry`.
`frDlcmiState`	Indicates which data link connection management interface (DLCMI) scheme is active (and by implication, what DLCI it uses) on the frame relay interface.
`frDlcmiAddress`	States which address format the frame relay interface is using.
`frDlcmiAddressLen`	States the address length in octets.
`frDlcmiPollingInterval`	Number of seconds between successive status-enquiry messages.
`frDlcmiFullEnquiryInterval`	Number of status-enquiry intervals that pass before a full status-enquiry message is issued.
`frDlcmiErrorThreshold`	Maximum number of unanswered status enquiries the equipment accepts before declaring the interface down.
`frDlcmiMonitoredEvents`	Number of status-polling intervals over which the error threshold is counted.

Table 19 Frame Relay Object Descriptions (*continued*)

Object Name	Description
frDlcmiMaxSupportedVCs	Maximum number of virtual circuits allowed for this interface.
frDlcmiMulticast	Indicates whether the frame relay interface is using a multicast service.
frDlcmiStatus	Indicates the status of the frame relay interface as determined by the performance of the DLCMI. If no DLCMI is running, the frame relay interface remains in the running state indefinitely.

Table 20 shows the frame relay OIDs and sample values. The frDlcmiIfIndex sample value indicates that frame relays are located on the interface at index 14. The frDlcmiAddress shows that the frame relay interface is using a 10-bit DLCI.

Table 20 Frame Relay OIDs and Sample Values

Object name	OID	Sample value
frDlcmiIfIndex	1.3.6.1.2.1.10.32.1.1.1.14	14
frDlcmiState	1.3.6.1.2.1.10.32.1.1.2.14	ansiT1617B
frDlcmiAddress	1.3.6.1.2.1.10.32.1.1.3.14	q922November90
frDlcmiAddressLen	1.3.6.1.2.1.10.32.1.1.4.14	twoOctets
frDlcmiPollingInterval	1.3.6.1.2.1.10.32.1.1.5.14	10
frDlcmiFullEnquiryInterval	1.3.6.1.2.1.10.32.1.1.6.14	6
frDlcmiErrorThreshold	1.3.6.1.2.1.10.32.1.1.7.14	3
frDlcmiMonitoredEvents	1.3.6.1.2.1.10.32.1.1.8.14	4

Table 20 Frame Relay OIDs and Sample Values

Object name	OID	Sample value
frDlcmiMaxSupportedVCs	1.3.6.1.2.1.10.32.1.1.9.14	992
frDlcmiMulticast	1.3.6.1.2.1.10.32.1.1.10.14	broadcast
frDlcmiStatus	1.3.6.1.2.1.10.32.1.1.11.14	running

The frCircuitLastTimeChange object indicates the value of sysUpTime (in seconds) when the last change occurred in the virtual circuit state. The frCircuitSentFrames object counts the number of frames sent from this virtual circuit since it was created. The frCircuitSentOctets object represents the number of octets counted for the full frame relay header and the payload, but does not include the flag characters or cyclic redundancy code (CRC).

For each frame relay interface, multiple circuits can exist. The next step is to identify the circuits and their respective interface index. The frCircuitIfIndex.?.? object is the ifIndex value of the ifEntry on which this virtual circuit is layered. The frCircuitDlci.?.? object indicates the DLCI for this virtual circuit. The frCircuitState.?.? object indicates whether this virtual circuit is operational. In the absence of a DLCI, you can create virtual circuit entries (rows) by setting the virtual circuit state to *active*, or delete them by changing circuit state to *invalid*. Whether or not the row actually disappears is left to the implementation, so this object may actually read as *invalid* for some arbitrary length of time. You can also set the state of a virtual circuit to *inactive* to temporarily disable a given circuit.

Note
The example in Table 21 has two frame relay circuits. The first circuit exists on interface index 14 and circuit DLCI 16; the second circuit exists on interface index 15 and circuit DLCI 20.

Table 21 Frame Relay Circuit Objects

Object name	OID	Sample value	Description
`frCircuitIfIndex.?.?`	1.3.6.1.2.1.10.32.2.1.1.14.16	14	`ifIndex` value of the `ifEntry` this virtual circuit is layered onto.
	1.3.6.1.2.1.10.32.2.1.1.15.20	15	
`frCircuitDlci.?.?`	1.3.6.1.2.1.10.32.2.1.2.14.16	16	DLCI for this virtual circuit.
	1.3.6.1.2.1.10.32.2.1.2.15.20	20	
`frCircuitState.?.?`	1.3.6.1.2.1.10.32.2.1.3.14.16	active	Indicates whether the particular virtual circuit is operational.
	1.3.6.1.2.1.10.32.2.1.3.15.20	active	

For each circuit, you can characterize the inbound and outbound traffic. The `frCircuitSentFrames.?.?` object indicates the number of frames sent from this virtual circuit since it was created; the `frCircuitSentOctets.?.?` object indicates the number of octets sent from this virtual circuit since it was created. This object counts full frame relay header and the payload octets but does not count the flag characters or the CRC.

The `frCircuitReceivedFrames.?.?` object indicates the number of frames received over this virtual circuit since it was created. The `frCircuitReceivedOctets.?.?` indicates the number of octets received over

this virtual circuit since it was created. This object counts the full frame relay header octets but not the flag characters or the CRC.

Table 22 Frame Relay Traffic Objects

Object name	OID	Sample value	Description
frCircuitSentFrames.?.?	1.3.6.1.2.1.10.32.2.1.6.14.16	0	Number of frames sent since virtual circuit creation.
	1.3.6.1.2.1.10.32.2.1.6.15.20	302	
frCircuitSentOctets.?.?	1.3.6.1.2.1.10.32.2.1.7.14.16	0	Number of octets sent since virtual circuit creation.
	1.3.6.1.2.1.10.32.2.1.7.15.20	25188	
frCircuitReceivedFrames.?.?	1.3.6.1.2.1.10.32.2.1.8.14.16	0	Number of frames received since virtual circuit creation.
	1.3.6.1.2.1.10.32.2.1.8.15.20	302	
frCircuitReceivedOctets.?.?	1.3.6.1.2.1.10.32.2.1.9.14.16	0	Number of octets received since virtual circuit creation.
	1.3.6.1.2.1.10.32.2.1.9.15.20	25188	

Additionally, you can characterize the circuit creation and change times of the frame relay circuit. The `frCircuitLastTimeChange.?.?` object contains the value of `sysUpTime` when the virtual circuit state was last changed.

Table 23 Frame Relay Circuit Property Objects

Object name	OID	Sample value	Description
`frCircuitLastTimeChange.?.?`	1.3.6.1.2.1.10.31.2.1.11.14.16	Wed Jul 5 20:45:30 2001	Value of `sysUpTime` when the virtual circuit state was last changed.
	1.3.6.1.2.1.10.31.2.1.11.15.20	Sat Jul 8 16:31:32 2001	

Finally, you can gather the static characteristics of each virtual circuit. The `frCircuitCommittedBurst.?.?` variable indicates the maximum amount of data, in bits, that the network agrees to transfer under normal conditions, during the measurement interval. The `frCircuitExcessBurst.?.?` variable indicates the maximum amount of uncommitted data bits that the network will attempt to deliver over the measurement interval. By default, if not configured when creating the entry, the excess information burst size is set to the value of `ifSpeed`.

Throughput, as indicated by the `frCircuitThroughput.?.?` object, is the average number of *Frame Relay Information Field* bits transferred per second across a user network interface in one direction, measured over the measurement interval. If the configured committed burst rate and throughput are both non-zero, the measurement interval, T, is:

T=`frCircuitCommittedBurst`/`frCircuitThroughput`

If the configured committed burst rate and throughput are both zero, the measurement interval, T, is:

T=`frCircuitExcessBurst`/`ifSpeed`

The `frCircuitMulticast.?.?` object indicates whether this virtual circuit is used as a unicast virtual circuit or, if multicast, the type of multicast service subscription the virtual circuit has.

The `frCircuitType.?.?` object provides an indication of whether the virtual circuit was manually created (static), or dynamically created (dynamic) with the data link control-management interface.

Table 24 Frame Relay Virtual Circuit Object Descriptions

Object	Description
`frCircuitCommittedBurst.?.?`	Maximum amount of data the network agrees to transfer under normal conditions.
`frCircuitExcessBurst.?.?`	Maximum amount of uncommitted data bits the network attempts to deliver.
`frCircuitThroughput.?.?`	One-way throughput across a user network interface.
`frCircuitMulticast.?.?`	Indicates whether this virtual circuit is unicast, or if multicast, the type of multicast service subscription it has.
`frCircuitType.?.?`	Indicates whether this virtual circuit is a manual or dynamic circuit.

Table 25 lists the frame relay OIDs and sample values.

Table 25 Frame Relay Virtual Circuit OIDs and Sample Values

Object name	OID	Sample value
`frCircuitCommittedBurst.?.?`	1.3.6.1.2.1.10.31.2.1.12.14.16	0
	1.3.6.1.2.1.10.31.2.1.12.15.20	0
`frCircuitExcessBurst.?.?`	1.3.6.1.2.1.10.31.2.1.13.14.16	2048000
	1.3.6.1.2.1.10.31.2.1.13.15.20	2048000

Table 25 Frame Relay Virtual Circuit OIDs and Sample Values (*continued*)

Object name	OID	Sample value
frCircuitThroughput.?.?	1.3.6.1.2.1.10.31.2.1.14.14.16	0
	1.3.6.1.2.1.10.31.2.1.14.15.20	0
frCircuitMulticast.?.?	1.3.6.1.2.1.10.31.2.1.15.14.16	unicast
	1.3.6.1.2.1.10.31.2.1.15.15.20	unicast
frCircuitType.?.?	1.3.6.1.2.1.10.31.2.1.16.14.16	static
	1.3.6.1.2.1.10.31.2.1.16.15.20	static

4.2.6 SNMP Group

The objects in this section are defined in RFC 1213 and RFC 1907.

This section discuss Objects that let you track the SNMP traffic of the managed network. Monitoring SNMP traffic helps you understand the activity of the management data.

The information in this section enables you to fully characterize all management traffic. Some objects provide extraneous and little-needed data. The complete set of objects is provided for future reference.

Table 26 SNMP Message Count Objects

Object name	OID	Sample value	Description
snmpInPkts	1.3.6.1.2.1.11.1.0	2731	Number of messages delivered to an SNMP entity from the transport service.
snmpOutPkts	1.3.6.1.2.1.11.2.0	2731	Number of SNMP Messages passed from an SNMP protocol entity to the transport service.

Table 27 SNMP Trap PDU Objects

Object name	OID	Sample value	Description
`snmpInTraps`	1.3.6.1.2.1.11.19.0	0	Number of SNMP trap PDUs (protocol data units) accepted and processed by the SNMP protocol entity.
`snmpOutTraps`	1.3.6.1.2.1.11.29.0	0	Number of SNMP trap PDUs generated by the SNMP protocol entity.

Table 28 SNMP Get and Set Objects

Object name	OID	Sample value	Description
`snmpInTotalReqVars`	1.3.6.1.2.1.11.13.0	18700	Number of MIB objects successfully retrieved by the SNMP protocol entity as the result of receiving valid SNMP `Get-Request` and `Get-Next` PDUs.
`snmpInTotalSetVars`	1.3.6.1.2.1.11.14.0	0	Number of MIB objects successfully altered by the SNMP protocol entity as the result of receiving valid SNMP `Set-Request` PDUs.
`snmpInGetRequests`	1.3.6.1.2.1.11.15.0	0	Number of SNMP `Get-Request` PDUs accepted and processed by the SNMP protocol entity.
`snmpOutGetResponses`	1.3.6.1.2.1.11.28.0	12116	Number of SNMP `Get-Request` PDUs generated by the SNMP protocol entity.
`snmpInGetResponses`	1.3.6.1.2.1.11.18.0	0	Number of SNMP `Get-Response` PDUs accepted and processed by the SNMP protocol entity.

Table 28 SNMP Get and Set Objects (*continued*)

Object name	OID	Sample value	Description
snmpOutGetRequests	1.3.6.1.2.1.11.25.0	0	Number of SNMP Get-Response PDUs generated by the SNMP protocol entity.
snmpInGetNexts	1.3.6.1.2.1.11.16.0	2741	Number of SNMP Get-Next PDUs which accepted and processed by the SNMP protocol entity.
snmpOutGetNexts	1.3.6.1.2.1.11.26.0	0	Number of SNMP Get-Next PDUs generated by the SNMP protocol entity.
snmpInSetRequests	1.3.6.1.2.1.11.17.0	0	Number of SNMP Set-Request PDUs accepted and processed by this SNMP protocol entity.
snmpOutSetRequests	1.3.6.1.2.1.11.27.0	0	Number of SNMP Set-Request PDUs generated by this SNMP protocol entity.

You can collect additional SNMP related values for error information, but these values are not important for performance monitoring.

Table 29 Additional SNMP Objects

Object name	OID	Sample value	Description
snmpInBadVersions	1.3.6.1.2.1.11.3.0	0	Number of SNMP PDUs delivered to this SNMP protocol entity with an error-status field value of badValue.
snmpInBadCommunityNames	1.3.6.1.2.1.11.4.0	0	Number of SNMP messages delivered to this SNMP entity that used an SNMP community name not known to this appliance.

Table 29 Additional SNMP Objects (*continued*)

Object name	OID	Sample value	Description
snmpInBadCommunityUses	1.3.6.1.2.1.11.5.0	0	Number of SNMP messages delivered to this SNMP entity that represented an SNMP operation not allowed by the SNMP community named in the message.
snmpInTooBigs	1.3.6.1.2.1.11.8.0	0	Number of SNMP PDUs delivered to this SNMP protocol entity with an error-status field value of tooBig.
snmpOutTooBigs	1.3.6.1.2.1.11.20.0	0	Number of SNMP PDUs generated by this SNMP protocol entity with an error-status field value of tooBig.
snmpInBadvalues	1.3.6.1.2.1.11.10.0	0	Number of SNMP PDUs delivered to this SNMP protocol entity with an error-status field value of badValue.
snmpOutBadValues	1.3.6.1.2.1.11.22.0	0	Number of SNMP PDUs generated by this SNMP protocol entity with an error-status field value of badValue.
snmpInNoSuchNames	1.3.6.1.2.1.11.9.0	0	Number of SNMP PDUs delivered to this SNMP protocol entity with an error-status field value of noSuchName.
snmpOutNoSuchNames	1.3.6.1.2.1.11.21.0	0	Number of SNMP PDUs generated by this SNMP protocol entity with an error-status field value of noSuchName.

Table 29 Additional SNMP Objects (*continued*)

Object name	OID	Sample value	Description
snmpOutGenErrs	1.3.6.1.2.1.11.24.0	0	Number of SNMP PDUs generated by the SNMP protocol entity with an error-status field value of genErr.
snmpEnableAuthenTraps	1.3.6.1.2.1.11.30.0	1	Indicates whether this SNMP entity is permitted to generate authenticationFailure traps.

Note
All SNMP traps need to be enabled using the Voyager interface. The value of the snmpEnableAuthenTraps object overrides any configuration information. You can use it to disable all authenticationFailure traps.

5.0 Summary

The SNMP MIB objects in this document enable you to precisely monitor and describe all network traffic that your Nokia appliance handles. You can use any network performance management application to measure and track the network performance of any Nokia appliance.

The objects in this analysis are not specific to Nokia platforms but are part of the standard SNMPv2 MIB, which means that you can use these objects on any network appliance.

6.0 References

This section lists the RFCs that correspond to the MIBs discussed in this document. You can access all of the RFCs listed below from the Internet Engineering Task Force Web site: http://www.ietf.org/rfc.html.

- SNMP v2 MIB (RFC 1907)
- SNMP v2 MIB (RFC 1213)
- IF (Interfaces) MIB (RFC 2233)
- IP MIB (RFC 2011)
- IP Forwarding (RFC 2096)
- TCP (RFC 2012)
- UDP (RFC 2013)
- Frame Relay DTE (RFC 2115)