DECpacketprobe 90

User's Information

Revision/Update Information: This is a new manual.

FCC NOTICE: The equipment described in this manual generates, uses, and may emit radio frequency energy. The equipment has been type tested and found to comply with the limits for a Class A computing device pursuant to Part 15 of FCC Rules, which are designed to provide reasonable protection against such radio frequency interference when operated in a commercial environment. Operation of this equipment in a residential area may cause interference, in which case the user at his own expense may be required to take measures to correct the interference.

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

3 Configuration

4 Problem Solving

Part III   RMON–MIB Reference

5 Remote Network Monitoring Management Information Base

6 Statistics Group

7 History Group

8 Alarm Group

9 Host Group

10 Host Top N Group

11 Matrix Group

12 Filter Group

13 Packet Capture Group

14 Event Group

A Specifications and Parts

B Documentation and Ordering Information

Index

Figures

Tables

About This Manual

Introduction

This manual describes how to install, configure, use, and troubleshoot the DECpacketprobe 90 agent module. The DECpacketprobe 90 implements the Ethernet Remote Network Monitoring Information Base (RMON–MIB). It can be used in a standalone operation or in the DEChub 90 or DEChub 900 Ethernet backplane.

Intended Audience

This manual is intended for experienced network managers.

Organization

This manual is organized as follows:

Part I, Product Overview

- Chapter 1 provides an overview of the DECpacketprobe 90. It also describes its features, LEDs, and connectors.

Part II, Installation and Configuration Procedures

- Chapter 2 describes how to install the DECpacketprobe 90 as a standalone unit or in a DEChub 90 or DEChub 900 backplane.

- Chapter 3 provides configuration information for correct operation of the DECpacketprobe 90.

- Chapter 4 discusses conditions that may affect DECpacketprobe 90 performance. It also contains basic troubleshooting information.

Part III, RMON–MIB Reference

- Chapter 5 discusses the Remote Network Monitoring Management Information Base (RMON–MIB).

- Chapter 6 contains the Statistics group parameters.

- Chapter 7 contains the History group parameters.

- Chapter 8 contains the Alarm group parameters.

- Chapter 9 contains the Host group parameters.

- Chapter 10 contains the Host Top N group parameters.

- Chapter 11 contains the Matrix group parameters.

- Chapter 12 contains the Filter group parameters.

- Chapter 13 contains the Packet Capture group parameters.

- Chapter 14 contains the Event group parameters.

Appendixes

- Appendix A contains DECpacketprobe 90 specifications and a parts list.

- Appendix B contains information about related documentation and ordering.

Part I

Product Overview

1
Overview

This chapter provides an overview of the DECpacketprobe 90 module. It lists the features of the module, describes the LEDs and connectors, and provides information about community strings.

1.1 Features

The following list summarizes the features of the DECpacketprobe 90:

- Implements the Remote Network Monitoring (RFC 1271) Management Information Base (MIB).

- Can be installed as a standalone unit *or* in a DEChub 90 or DEChub 900 backplane.

- Needs no downline load at powerup.

- Is downline loadable for firmware upgrades (TFTP protocol).

- Can be directly managed through SNMP.

- Stores MIB settings in battery-backed RAM.

- Provides an out-of-band console port.

- Allows hot-swapping of modules.

1.2 Description

The DECpacketprobe 90 module is a network management agent designed for IEEE 802.3 CSMA/CD networks. It uses Simple Network Management Protocol (SNMP) and has one each of the following ports:

- 10Base2 (ThinWire)

- 10BaseT (shielded or unshielded, twisted-pair)

- Asynchronous RS–232 (DB9)

The ThinWire port and the twisted-pair port comply with IEEE 802.3 standards.

The DECpacketprobe 90 is a self-contained module that uses a Motorola 68ECO30 central processing unit (CPU). The DECpacketprobe 90 can operate in a standalone configuration with a 5-volt power supply, or in a DEChub 90 or DEChub 900 Ethernet backplane.

Figure 1–1 shows a typical configuration of the DECpacketprobe 90 in an Ethernet local area network (LAN).

1.3 How DECpacketprobe 90 Works

DECpacketprobe 90 agents gather a wide variety of statistical information about network operation. An agent gathers the information by examining each packet passed on a network segment. Segment statistics are stored in counters within the agent. The counters are continuously updated and reset at powerup.

In addition to statistics, the agent captures and stores network traffic information. You can examine these individual packets or sequences of packets to identify and isolate network operational software or hardware problems.

A typical network consists of multiple network segments with one DECpacketprobe 90 connected to each segment. You can control the agent from a centrally located network management station designated as the "client." The client has a user interface that permits you to request and examine data from a selected agent. You can have multiple clients active to perform network diagnostic functions from multiple locations such as primary and secondary network management centers.

Figure 1–1   Sample DECpacketprobe 90 Configuration
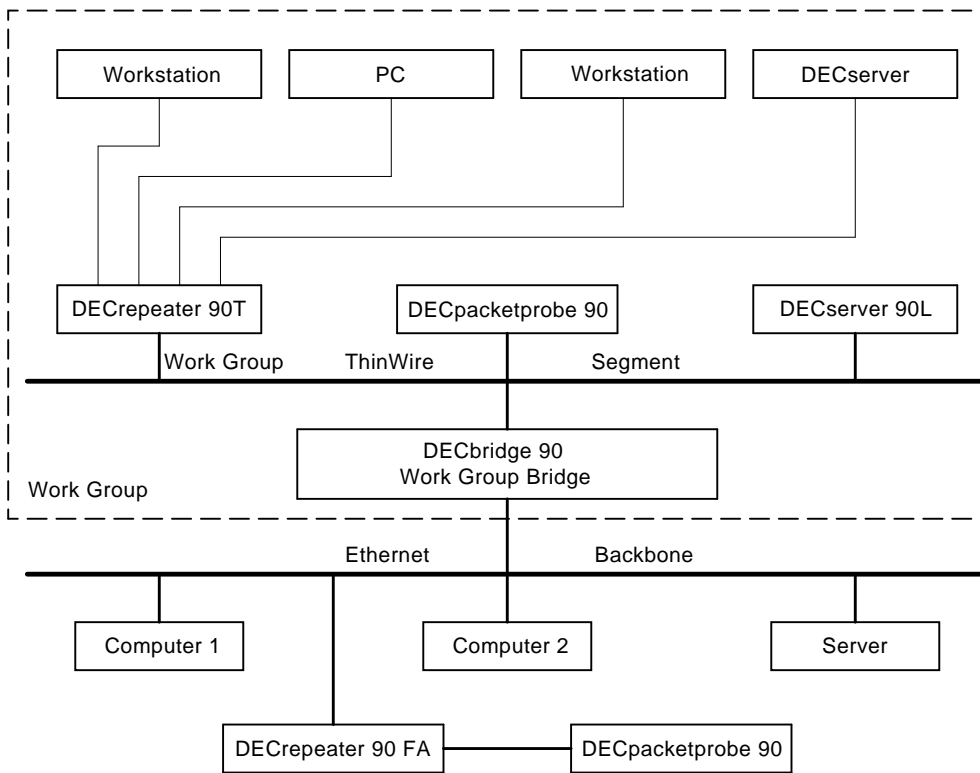
LJ-03390-TI0

1.4 Standards

DECpacketprobe 90 agents are based on widely accepted industry standards. Communications between agents and clients are performed using Simple Network Management Protocol (SNMP). The statistical information gathered is defined by the Remote Network Monitoring Management Information Base (RMON–MIB) as defined for Ethernet networks.

In addition to the Ethernet RMON–MIB groups (RFC 1271), DECpacketprobe 90 supports:

- MIBII System and Interface groups (RFC 1213)
- Enterprise-specific MIB extensions

DECpacketprobe 90 does not support the following MIBII groups:

- Address translation
- IP
- ICMP
- TCP
- UDP
- EGP
- Transmission
- SNMP

1.4.1 RMON–MIB

The RMON–MIB is a standard developed under the auspices of the Internet Engineering Task Force (IETF) as an extension of their pioneering work in the development of SNMP. This standard provides a data object base compatible with the widest range of network needs, and permits independent developers to design network monitoring equipment that interoperates.

Table 1–1 describes the nine RMON–MIB groups for Ethernet that DECpacketprobe 90 supports. A detailed description of the RMON–MIB and the Ethernet groups is in Part III of this manual.

Table 1–1 RMON–MIB Ethernet Groups

This group . . .	Allows you to . . .
Alarms	Set a wide variety of thresholds and sampling intervals on any statistic to create an alarm condition. You may set threshold values as: • An absolute value • A rising value • A falling value • A delta value so that each node or segment may be fully customized
Events	Create entries in the monitor log and generate SNMP traps from the agent to the client for selected events. You can initiate events by setting a crossed threshold on any counter or from a specific packet match count. The log includes the time of day for each event and a description of the event.
Filters	Define specific packet match filters that serve as a stop/start mechanism for all packet capture functions and events. You may combine filters with AND, NOT, or OR functions to capture either broad or unique network events.
History	Obtain a historical representation of statistics based on user-defined sample intervals and bucket counters for customized trend analysis.
Host Table	Gather information for all nodes retrievable under SNMP for non-SNMP devices by means of a host table. A table for each node includes a variety of node statistics for each active node, including the relative time the node was discovered.
HostTopN	Perform a user-defined study of sorted host statistics providing detailed information on the top n occurrences (where n is a number supplied by you). Performed locally by the agent, this function substantially reduces network traffic.
Packet Capture	Perform further analysis using matched packets captured and stored under the control of selected filters. Buffer sizes may be user-selected and may wrap or stop when full. You may upload captured packets to a centralized client that, if equipped with protocol decode software, will allow you to perform complete protocol analysis.

Table 1–1 (Cont.)   RMON–MIB Ethernet Groups

This group . . .	Allows you to . . .
Statistics	Obtain an array of operational statistics including the following: • Packets • Octets • Broadcasts • Collisions • Dropped packets • Fragments • CRCalignment errors • Undersize/oversize
Traffic Matrix	Obtain a matrix that shows the amount of traffic and the number of errors between any pair of nodes. Retrieved by either source or destination address.

1.4.2 Protocols

The DECpacketprobe 90 has a single IP address. It implements the protocols required for SNMP-based access. The following protocols are used:

• Simple Network Management Protocol (SNMP)

• Address Resolution Protocol (ARP)

• Internet Protocol (IP)

• User Datagram Protocol (UDP)

• Telnet File Transfer Protocol (TFTP)

_____ **Note** _____

TFTP is for network loading of upgrades.

_____

1.5 Community Strings for the DECpacketprobe 90

A community, in SNMP, is a set of manageable attributes that are managed as a group. Normally, there is a one-community-to-one-agent relationship. The manageable attributes are usually contained within a single hardware device, or within a single enclosure, when referenced with hubs. The single hardware device, or the collection of devices within a hub, is treated as one community. A particular manageable entity is uniquely identified on the network by the combination of an IP address and a community string.

A community string is a sequence of ASCII characters that is checked by the SNMP agent for access control to the manageable entity. The community string can be thought of as a password. Two strings are associated with a given community: the read-only string and the read/write string. For a GET or a GET NEXT operation, the agent accepts either the read-only or the read/write string. However, for a SET operation, the agent accepts only the read/write string.

Each community for a DECpacketprobe 90 is identified by a unique community address string for each standalone unit or module installed in a DEChub 90 or 900 backplane.

The easiest way to keep track of the modules with community address strings is to use a similar name for the DECpacketprobe 90 and community address string. The community string name can be up to 32 characters. A community string consists of any printable characters and is case-sensitive. Here are two examples of naming schemes:

```
MyHUBFloor3Room27
```

```
COMM_LAB_HUB_3
```

For more information about communities, see Chapter 5.

1.6 LEDs and Connectors

Figure 1–2 shows the DECpacketprobe 90 and Table 1–2 describes its LEDs and connectors.

Figure 1–2 The DECpacketprobe 90 Module
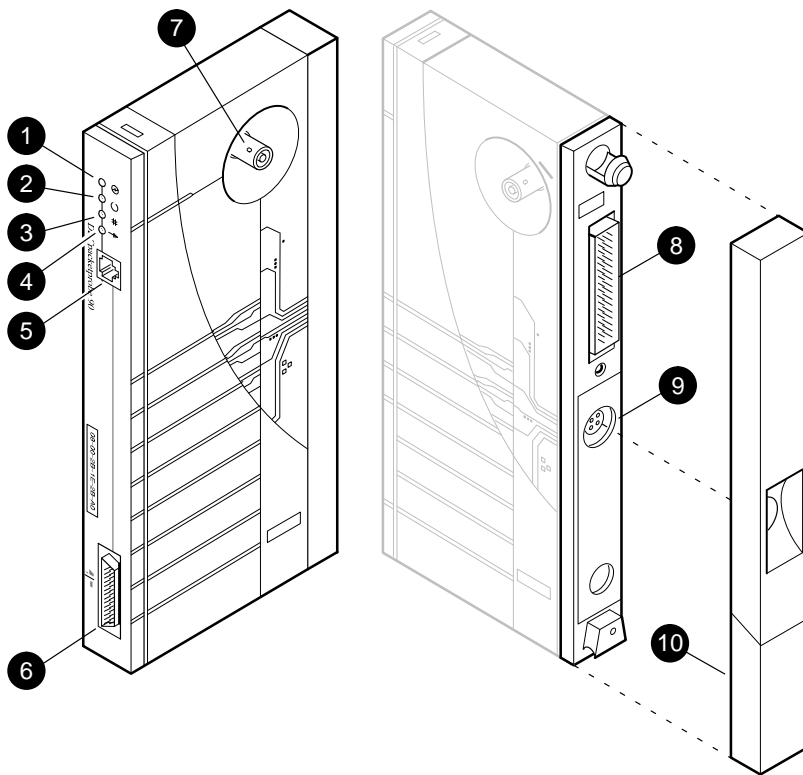
LKG-8011-93I

Table 1–2   DECpacketprobe 90 LEDs and Connectors

Item	Icon	Description
❶	⊘	**DC OK indicator** — Monitors the voltage. **On:** The +5.0 Vdc voltage is normal. **Off:** The voltage is abnormal (as when a power failure occurs).
❷	◯	**Network activity indicator** — Indicates network activity. **Flashing:** Network activity. **Off:** No network activity.
❸	⌗	**Network OK indicator** — Monitors the network port. **On:** The network port is properly terminated. **Off:** The network port is not properly terminated.
❹	→	Not used.
❺		**10BaseT Ethernet connector** — Connects the agent to the twisted-pair segment. Not used when the agent is installed in the DEChub 90 or 900 backplane.
❻		**Asynchronous serial port (RS–232) connector** — Provides an interface to the DECpacketprobe 90 through a console terminal or terminal emulation program.
❼		**Port 0 network connector (BNC)** — Connects the agent to the ThinWire segment. Not used when the agent is installed in the DEChub 90 or 900 backplane.
❽		**Backplane connector** — Provides network and power connections to the DECpacketprobe 90 when it is installed in the DEChub 90 or 900 backplane.
❾		**Power connector** — Receives +5.0 volts from the DECpacketprobe 90 power supply. Not used when the agent is installed in the DEChub 90 or 900 backplane.
❿		**Back cover** — Used with standalone units only. It covers the backplane connector and mounting assembly.

Part II

Installation and Configuration Procedures

2

Installation

This chapter describes how to install the DECpacketprobe 90 as a standalone unit, or in a DEChub 90 or DEChub 900 backplane.

2.1 Inventory

You should have received the following items:

- 1 DECpacketprobe 90

- 2 mounting screws

- 1 power supply and ac power cord (if the DECpacketprobe 90 is ordered for standalone use)

2.2 Standalone Installation

The DECpacketprobe 90 can be installed either on a tabletop or on a wall. The location should be within 2 meters (6 feet) of a power outlet that supplies the correct voltage (Appendix A).

When operating as a standalone unit, the DECpacketprobe 90 receives power from a separate power supply. This power supply connects to the unit by a cable with a 7-pin connector. There is one universal autosensing power supply with power cords appropriate to local convention.

2.2.1 Tabletop Installation

For tabletop installation, place the DECpacketprobe 90 on a table and connect the cables as described in Section 2.2.2.

2.2.2 Wallmount Installation

To install the DECpacketprobe 90 on a wall:

1. Remove the back cover (Figure 2–1).

    a. Insert a small screwdriver into the top mounting hole on the cover.

    b. Lift the latch with the screwdriver.

    c. Pull the cover away and down from the top of the unit.

Figure 2–1 Removing the Back Cover
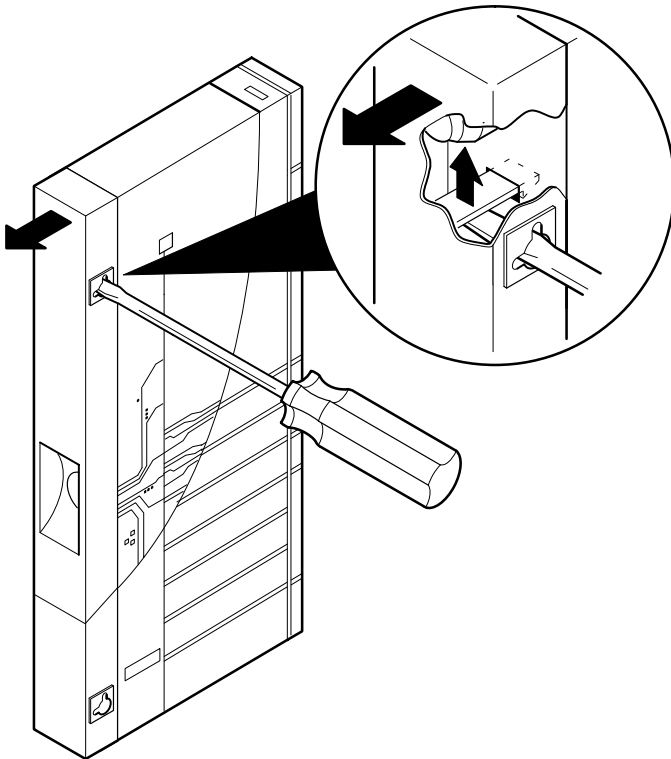
LKG-8010-93I

2. Use the mounting holes on the back cover (Figure 2–2, ❶) to determine the placement for the mounting screws on the wall.

3. Secure the back cover to the wall using the mounting screws (Figure 2–2, ❷).

   The screws should be tight enough to provide resistance when you remove the cover from the wall. Do not make them so tight that the cover is distorted or cannot be removed from the wall.

Figure 2–2 Mounting Hole Locations
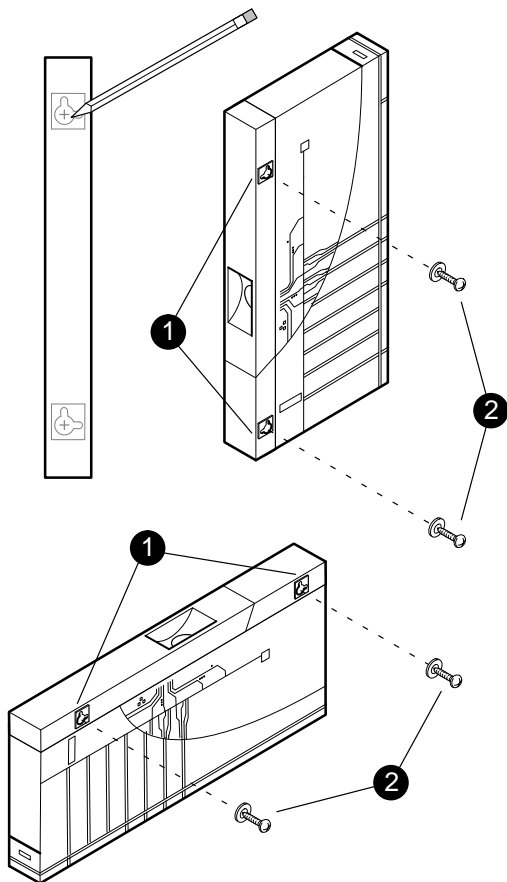
LKG-8015-93I

4. Remove the back cover from the wall, attach the cover to the unit, then mount the unit onto the mounting screws.

5. Connect the cables (Figure 2–3).

   a. Connect the Ethernet ThinWire T-connector to the port 0 BNC network connector ❶.

      or

      Connect the twisted-pair connector as shown ❷.

   b. Connect the cable from the power supply (H7827–BA) to the 7-pin power connector on the DECpacketprobe 90 ❸.

   c. Connect a terminal to the front panel asynchronous port ❹. This connection allows console management.

Figure 2–3 Connecting the Cables
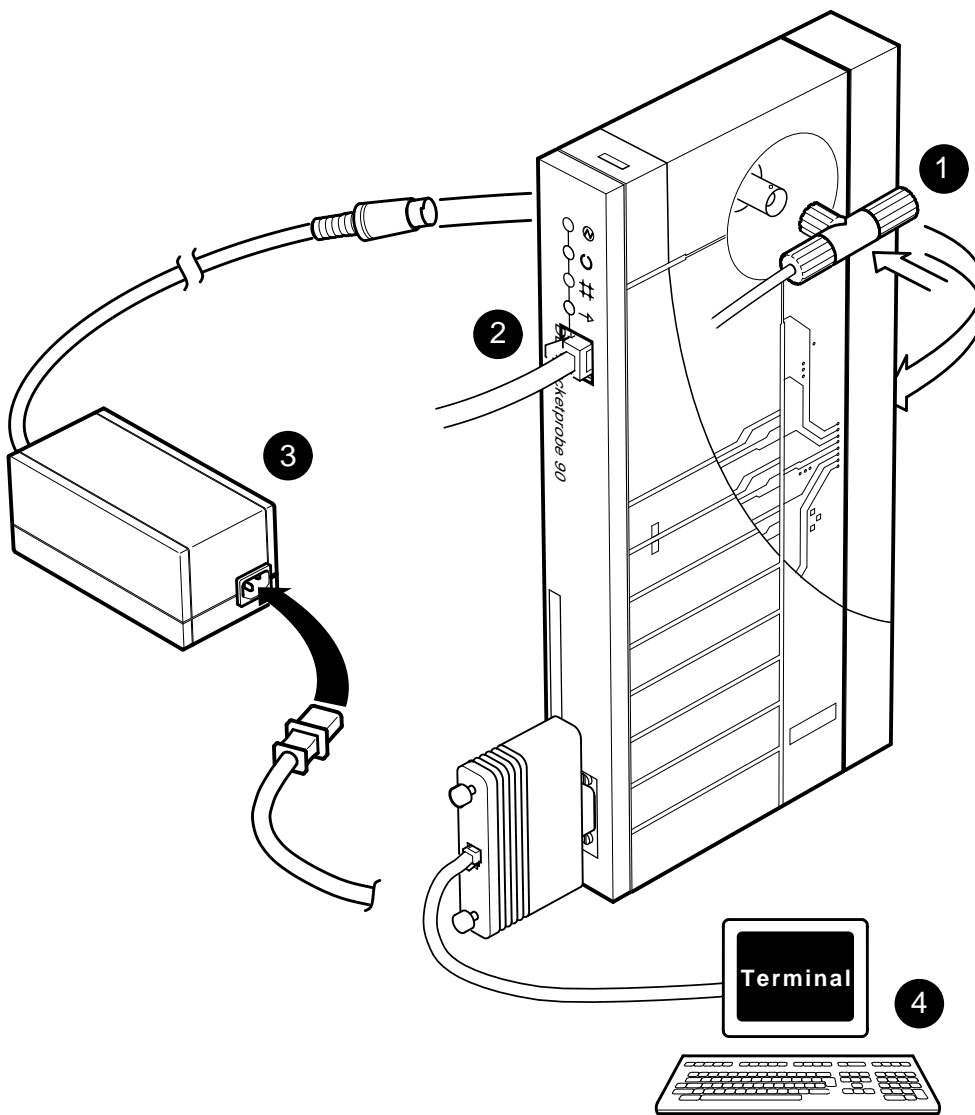
LJ-03389-TI0

2.3 Backplane Installation

To install the DECpacketprobe 90 in a DEChub 90 or 900 Ethernet backplane:

1. Remove the back cover (Figure 2–1).

2. Install the DECpacketprobe 90 in the backplane (Figure 2–4). Place the lower mounting tab, located on the back of the DECpacketprobe 90, into any slot of the backplane.

3. Rock the unit until it clicks into place.

_____ **Note** _____

The DECpacketprobe 90 can be installed into or removed from the DEChub 90 or 900 Ethernet backplane while power is supplied to the backplane. This procedure is referred to as a "hot swap."

_____

Figure 2–4 DECpacketprobe 90 Backplane Installation
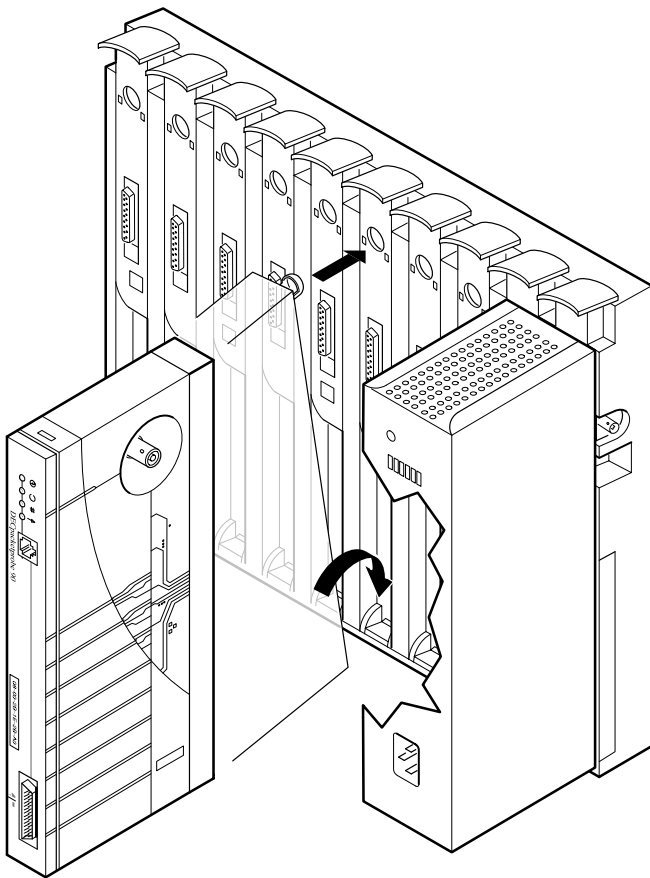
LKG-8014-93I

2.4 Powerup and Reset

Upon initial powerup of the DECpacketprobe 90, the DC OK LED lights. After approximately 5 seconds, the network activity LED lights with intensity that is proportional to the amount of network activity on the port.

Upon powerup and reset, the DECpacketprobe 90 performs self-test out of ROM. The unit enters an operational mode if an IP address has been established. If an IP address has not been established, the unit does not attempt to obtain its IP address. You can establish one with a terminal through the console port.

If the unit encounters no FLASH image or a corrupted image during powerup or reset, the unit enters the monitor mode. While the unit is in monitor mode, you can access error information and logs through the console port.

To verify the operating mode, start a console sesion. The console screen indicates whether the unit is operating from FLASH nonvolatile RAM (NVRAM) or EPROM. The DECpacketprobe 90 should always operate out of FLASH NVRAM with one exception, during a FLASH update.

See Chapter 3 for information about starting a console session to set an IP address, access error information, or check the operating mode.

3
Configuration

The DECpacketprobe 90 agent software starts to run upon initial installation. At this point, you must initialize the following agent parameters:

- IP address
- Net mask
- Gateway address

The procedure is done from the console. During the console session, you can also access and change some of the read/write parameters of the agent.

3.1 Console Session

Since the module comes preloaded with the agent software, you may initiate a session with the agent through use of the agent console port. To start a console session, connect a terminal to the console port on the DECpacketprobe 90 and press ⟨Return⟩ twice.

The following screen is displayed:

```
        ****** DEC PacketProbe 90 Ethernet Probe Rev 1.0 ******

                        Executing from FLASH

[1]  Change IP Address
[2]  Change Net Mask
[3]  Change Default Gateway Address
[4]  Change Read Community
[5]  Change Write Community
[9]  Change TFTP Server Address
[10] Update Software
[11] Enter Command-line Mode
[12] Reset Agent

Selection#:
```

- **Selections 1 through 9** allow you to view or change the corresponding information. When you choose one of these selections, the current state of the parameter is displayed. To change the state of the parameter, enter the new information and press Return . To make no change, simply press Return .

- **Selection 10** allows you to update the firmware in FLASH NVRAM. To be updated, the DECpacketprobe 90 must be connected to a TFTP server and the server must have the update image loaded.

- **Selection 11** allows you to enter the command-line mode to access or change agent parameters. Table 3–1 describes the permissible actions in command-line mode, and Table 3–2 describes each object variable that you can associate with the actions.

  Entering **help** at the command-line prompt (%) displays the following help screen:

```
%help

      Syntax: action object [arg,...]
actions
   get: display the value of an object
   set: change the value of an object
    do: perform an agent operation
  help: display this menu
  quit: exit command-line mode

objects
 agent          agent_contact     agent_location     agent_name
 agent_options  eventlog          gw_addr            if_options
 interface      ip_addr           max_captsize       max_host
 max_log        max_matrix        myu_size           net_mask
 nvram          read_community    reset              speed
 tftp           tftp_timeout      tftp_server        write_community

%
```

- **Selection 12** allows you to reset the agent and saves the current agent parameters in battery-backed RAM.

Table 3–1 Console Actions

Action	Description
get	Retrieves the value of the designated object.
set	Sets the value of the designated object.
do	Produces an action in the agent.
help	Provides a list of permissible commands and command objects.
quit	Exits from the command-line mode.

Table 3–2 Object Variables

Variable	Description
agent	A composite list of the agent parameters and descriptions.
agent_contact	An ASCII text string corresponding to the MIBII system group variable **sysContact**.
agent_location	An ASCII text string corresponding to the MIBII system group variable **sysLocation**.
agent_name	An ASCII text string corresponding to the MIBII system group variable **sysName**.
agent_options	The object variables that can be obtained through command-line mode.
eventlog	A log maintained at the agent. Includes a list of the most recent events that have occurred at the agent. These events are stored as they occur.
gw_addr	Gateway address.
if_options	Not currently used.
interface	IP address, gateway address, and net mask address.
ip_addr	IP address.
max_captsize	Not currently used.
max_host	Decimal value specifying the maximum number of entries in each instance of the RMON–MIB hostTable and hostTimeTable. The agent removes the least recently seen host entries when this number is exceeded. The default value is 500.
max_log	Decimal value specifying the maximum number of entries in the RMON–MIB logTable. The default value is 500.

(continued on next page)

Table 3–2 (Cont.)   Object Variables

Variable	Description
max_matrix	Decimal value specifying the maximum number of entries in each instance of the RMON–MIB matrixDSTable and matrixSDTable. The agent removes the least recently seen matrix entries when this number is exceeded. The default value is 4000.
myu_size	Not currently used.
net_mask	Address for a network structure.
nvram	Not currently used.
read_community	Read community string.
reset	Agent reset.
speed	Baud rate.
tftp	Utility for agent updates.
tftp_timeout	Value specifying the response timeout from the TFTP server.
tftp_server	Address for the TFTP server.
write_community	Write community string.

4

Problem Solving

This chapter describes the conditions that influence the performance of the DECpacketprobe 90. It also provides information for solving problems related to the DECpacketprobe 90 when used as a standalone unit, or when installed in a DEChub 90 or 900 backplane.

4.1 Performance Considerations

Protocol monitoring and analysis is a real-time function. Every packet and event on the segment being monitored by the DECpacketprobe 90 must be examined, evaluated, and counted. In some cases, a sequence of filters must be applied to the data before specific action such as counts or data storage take place. In addition, irregular events such as packet fragments, jabbers, and collisions must be recognized, counted, and discarded.

The capability of the DECpacketprobe 90 to process all packets and handle all other requirements in real-time depends on many factors. The most common factors are the:

- Fundamental speed of the processor

- Arrival rate of data and events on the segment

- Amount of work the agent has been configured to do by the client

- Amount of processor time that must be devoted to other concurrently running programs

Predicting the specific performance of an agent is impossible due to the great number of variables in a normal network configuration.

At some point, the agent will not be able to respond as required to all the network traffic. An example is when one agent is performing different functions under the direction of two or more clients. Under these conditions, the following may occur:

- Response to user inquiries from the client may slow down.

- Some packets will be missed. The number of occasions when one or more packets is missed is in the dropEvents statistic.

4.2 Troubleshooting the DECpacketprobe 90

To determine the cause of a problem with the DECpacketprobe 90:

1. Verify the installation of the DECpacketprobe 90.

2. Verify the IP, net mask, and gateway addresses.

   Installation must meet the configuration information given in Chapter 3.

3. Note the fault condition.

4. Verify that the DECpacketprobe 90 is operating from FLASH NVRAM.

5. Isolate the problem.

   - Use Table 4–1 to solve problems related to a standalone DECpacketprobe 90.

   - Use Table 4–2 to solve problems related to a DECpacketprobe 90 installed in a DEChub 90 or 900 backplane.

Table 4–1 Problem Solving a DECpacketprobe 90 Standalone Unit

If . . .	Then . . .	Do this . . .
All LEDs are off.	Check the ac power connection.	Check that the power supply is properly plugged into the power outlet. Check the power to the power outlet.
	Check the power supply.	Check the 7-pin connector to the power supply. If the connections are okay, replace either the power supply (for 120 V operation) or both the power supply cord and power supply (for 220 V operation).

(continued on next page)

Table 4–1 (Cont.)   Problem Solving a DECpacketprobe 90 Standalone Unit

If . . .	Then . . .	Do this . . .
All LEDs flicker.	The DECpacketprobe 90 is not receiving the correct voltages.	Connect the DECpacketprobe 90 to the correct power supply or replace the power supply.
The network activity LED is off.	There is low network activity or no network activity.	Ensure that network activity is present and that the segment is properly terminated.
	If the network activity LED still fails to light	Turn the agent off and on by unplugging and replugging the power supply. Check that the network activity LED comes on momentarily.
	If the network activity LED lights momentarily, the network activity LED portion of testing has passed.	Replace the DECpacketprobe 90.
	If the network activity LED fails to light.	Replace the DECpacketprobe 90.

Table 4–2   Problem Solving a DECpacketprobe 90 in a DEChub 90 or 900 Backplane

If . . .	Then . . .	Do this . . .
The power LED is off.	The DECpacketprobe 90 is not receiving +5 V.	Check the power LED on the DEChub 90 or 900 power supply.
	If the power LED on the DEChub 90 or 900 is off, there is a problem with the DEChub 90 or 900 power supply.	See the problem solving procedures in the owner's manual for the DEChub 90 or DEChub 900.
	If the power LED on the DEChub 90 or 900 power supply is on, are other component power LEDs off? If other components are on, the DECpacketprobe 90 has a power problem.	Try reseating the DECpacketprobe 90 in either the same slot or the other slot.

Table 4–2 (Cont.) Problem Solving a DECpacketprobe 90 in a DEChub 90 or 900 Backplane

If . . .	Then . . .	Do this . . .
	If the power LED on the DEChub 90 or 900 power supply is on, are other component power LEDs off? If other components are off, the DEChub 90 or 900 has a power problem.	See the problem solving procedures in the owner's manual for the DEChub 90 or DEChub 900.
	If the power LED turns on when reseated in the same slot, the DECpacketprobe 90 was not properly seated.	Make sure that the DECpacketprobe 90 is properly seated in the slot.
	If the power LED turns on when reseated in another slot, the problem is with the DEChub 90 or 900.	See the problem solving procedures in the owner's manual for the DEChub 90 or DEChub 900.
	If the DECpacketprobe 90 does not turn on in a known good slot, the DECpacketprobe 90 is defective.	Replace the DECpacketprobe 90.
The port 0 network activity LED is off.	The DECpacketprobe 90 or any other unit in the DEChub 90 or 900 may not be connected to an active segment.	Connect a known active segment to a port (other than port 0) on any unit in the DEChub 90 or 900.
	If the DECpacketprobe 90 is connected to a known active segment and the port 0 LED is off, the DECpacketprobe 90 is defective.	Replace the DECpacketprobe 90.

Part III

RMON–MIB Reference

5

Remote Network Monitoring Management Information Base

This chapter includes a detailed description of the functions and acknowledgment objects that have been defined by the Internet Engineering Task Force (IETF). This information was derived from a draft memo produced by the Remote Monitoring Working Group of the IETF dated November 1991, and that was submitted to the RFC editor as an experimental extension to the SNMP MIB.

5.1 Overview of SNMP and Dot Notation

Over the last few years, SNMP has become the protocol of choice for the management of internet work resources. As its name implies, it has the advantage of simplicity and great flexibility for a wide variety of applications. The addition of the RMON–MIB subgroup under SNMP has resulted in a standard framework for performing network diagnostics. For an overview of SNMP and its use in network management, see *The Simple Book: An Introduction to Management of TCP/IP-based Internets*, Marshall T. Rose, Prentice Hall, Englewood Cliffs, NJ, 1991.

SNMP is an application-layered protocol, running over UDP that sits on top of IP in the TCP/IP protocol stack. It was developed in the late 1980s by a committee of the Internet Activities Board and has undergone revisions and enhancements since that time. SNMP is a protocol that network management stations use to poll network devices that are equipped with "agents" that collect and store information in a Management Information Base (MIB). Agents send information to the network management station when requested and otherwise act on the direction of the centralized control.

SNMP has four operations:

- Get — Used to retrieve specific management information.
- Get-next — Used to retrieve management information relative to the argument included in the command.
- Set — Used to manipulate information and establish base values.
- Trap — Used to report specified events.

The above commands are performed on data objects that are uniquely identified by an Object Identifier. An Object Identifier is a sequence of decimal integer values where each value is separated by a "dot." Each decimal value, defined in Table 5–1, represents a branch along a decision tree where the final step may be considered a leaf. Therefore, each element of a MIB has its own unique address using dot notation and also a textual description.

The prefix for an Object Identifier for an RMON instance is

```
1.3.6.1.2.1.16.
```

Within RMON–MIB, the groups are identified as described in Table 5–2. Using the construction in Table 5–2, the Object Identifiers for a few typical objects are as follows:

```
1.3.6.1.2.1.16.1.1.1.3.1      etherStatsDropEvents.1
1.3.6.1.2.1.16.1.1.1.4.1      etherStatsOctets.1

1.3.6.1.2.1.16.1.1.1.20.1     etherStatsOwner.1
1.3.6.1.2.1.16.6.1.1.6.3      matrixControlStatus.3
```

The following chapters, derived from the RMON–MIB standard document, can be better understood in the context of the above description of the Object Identifier. Each variable in the MIB is shown with its relative position for the entry using the notation { xxxxxxEntry *n* }. For example:

```
etherStatsDropEvents is { etherStatsEntry 3 }
etherStatsOctets is { etherStatsEntry 4 }
matrixControlStatus is { matrixControEntry 6 }
```

In addition, the following chapters include the formal textual description for each object in the RMON–MIB.

Table 5–1 Decimal Definitions for the Object Identifier Prefix

Decimal Position	Description
1	iso (1) — International Standards Organization
2	identified-organization (3) — A defined organization as opposed to other standards groups
3	dod (6) — Controlling body for TCP/IP
4	internet (1) — For internet applications
5	management (2) — A network management function
6	mib_2 (1) — Management Information Base **Note:** MIB2 is a superset of MIB1.
7	rmon (16) — Remote monitoring

Table 5–2 Decimal Definitions for Groups in RMON–MIB

Decimal Position	Description
8	Consists of the following items: statistics (1) history (2) alarm (3) hosts (4) hostTopN (5) matrix (6) filter (7) capture (8) event (9) tokenRing (10)
9	Table (1) to (n) corresponding to control entries.
10	Entry (1) to (n) is the number of entries in the corresponding table.
11	Variable (1) to (n) depending on the group.
12	Instance (1) to (n) is a unique number for each instance of a control entry for this object.

5.2 The Network Management Framework

The Internet-Standard Network Management Framework consists of the following components:

- RFC 1155 defines the Structure of Management Information (SMI). These are the mechanisms used for describing and naming objects for the purpose of management. RFC 1212 defines a more concise description mechanism that is wholly consistent with the SMI.

- RFC 1156 defines MIBI. This is the core set of managed objects for the Internet suite of protocols. RFC 1213 defines MIBII, an evolution of MIBI-based on implementation experience and new operational requirements.

- RFC 1157 defines the SNMP. This is the protocol used for network access to managed objects. The Framework permits new objects to be defined for the purpose of experimentation and evaluation.

5.3 Managed Objects

Managed objects are accessed by way of a virtual information store, termed the Management Information Base, or MIB. Objects in the MIB are defined using the subset of Abstract Syntax Notation One (ASN.1) [7] defined in the SMI. In particular, each object has a name, a syntax, and an encoding.

The name is an object identifier, an administratively assigned name, that specifies an object type. The object type together with an object instance serves to uniquely identify a specific instance of the object. A textual string is used, termed the Object Descriptor, to also refer to the object type.

The syntax of an object type defines the abstract data structure corresponding to that object type. The ASN.1 language is used for this purpose. However, the SMI [3] purposely restricts the ASN.1 constructs that may be used. These restrictions are made for simplicity.

The encoding of an object type is how that object type is represented using the object type's syntax. Implicitly tied to the notion of an object type's syntax and encoding is how the object type is represented when being transmitted on the network.

The SMI specifies the use of the basic encoding rules of ASN.1 [8], subject to the additional requirements imposed by the SNMP. The object types contained in the RMON–MIB module are defined using the conventions defined in their SMI, as amended by the extensions specified in [9,10].

5.4 Overview of Remote Network Monitoring

Remote network monitoring devices are instruments that exist for the purpose of managing a network. Often, these remote probes are standalone devices and devote significant internal resources for the sole purpose of managing a network. An organization may employ many of these devices (one per network segment) to manage its internet. In addition, these devices may be used for a network management service provider to access a client network, often geographically remote.

5.4.1 Remote Network Monitoring Goals

Remote Network Monitoring goals include the following:

- Off-line Operation — Sometimes there are conditions when a management station will not be in constant contact with its remote monitoring devices. This is sometimes by design in an attempt to lower communications costs (especially when communicating over a WAN or dialup link), or by accident as network failures affect the communications between the management station and the probe.

  For this reason, this MIB allows a probe to be configured to perform diagnostics and to collect statistics continuously, even when communication with the management station may not be possible or efficient. The probe may then attempt to notify the management station when an exceptional condition occurs. Fault, performance, and configuration information may be continuously accumulated and communicated to the management station conveniently and efficiently, even in circumstances where communication between management station and probe is not continuous.

- Pre-emptive Monitoring — Given the resources available on the monitor, it is potentially helpful for it to run diagnostics continuously and to log network performance. The monitor is always available at the onset of any failure. It can notify the management station of the failure and can store historical statistic information about the failure. This historical information can be played back by the management station in an attempt to perform further diagnosis into the cause of the problem.

- Problem Detection and Reporting — The monitor can be configured to recognize conditions, most notably error conditions, and continuously to check for them. When one of these conditions occurs, the event may be logged, and management stations may be notified in a number of ways.

- Value Added Data — Because a remote monitoring device represents a network resource dedicated exclusively to network management functions, and because it is located directly on the monitored portion of the network, the remote network monitoring device has the opportunity to add significant value to the data it collects. For instance, by highlighting those hosts on the network that generate the most traffic or errors, the probe can give the management station precisely the information it needs to solve a class of problems.

- Multiple Managers — An organization may have multiple management stations for different units of the organization, for different functions (for example, engineering and operations), and in an attempt to provide disaster recovery. Because environments with multiple management stations are common, the remote network monitoring device has to deal with more than its own management station, potentially using its resources concurrently.

5.4.2 Structure of the RMON–MIB

The objects are arranged into the following groups:

- Statistics

- History

- Alarm

- Host

- HostTopN

- Matrix

- Filter

- Packet Capture

- Event

- Token Ring

These groups are the basic unit of conformance. If a remote monitoring device implements a group, it must implement all objects in that group. For example, a managed agent that implements the Host group must implement the hostControlTable, the hostTable, and the hostTimeTable.

All groups in this MIB are optional. Implementations of this MIB must also implement the System and Interface groups of MIBII [6]. MIBII may also mandate the implementation of additional groups.

These groups are defined to provide a means of assigning object identifiers, and to provide a method for managed agents to know what objects they must implement.

5.4.3 Control of Remote Network Monitoring Devices

Due to the complex nature of the available functions in these devices, the functions often need user configuration. In many cases, the function requires parameters to be set up for a data collection operation. The operation can proceed only after these parameters are fully set up.

Many functional groups in this MIB have one or more control tables in which to set up control parameters, and one or more data tables in which to place the results of the operation. The control tables are typically read-write in nature. The data tables are typically read-only. Because the parameters in the control table often describe resulting data in the data table, many of the parameters can be modified only when the control entry is invalid. Thus, the method for modifying these parameters is to invalidate the control entry, causing its deletion and the deletion of any associated data entries, and then to create a new control entry with the proper parameters. Deleting the control entry also gives a convenient method for reclaiming the resources used by the associated data.

Some objects in this MIB provide a mechanism to execute an action on the remote monitoring device. These objects may execute an action as a result of a change in the state of the object. For those objects in this MIB, a request to set an object to the same value as it currently holds would thus cause no action to occur.

To facilitate control by multiple managers, resources have to be shared among the managers. These resources are typically the memory and computation resources that a function requires.

5.4.4 Resource Sharing Among Multiple Management Stations

When multiple management stations want to use functions that compete for a finite amount of resources on a device, a method to facilitate this sharing of resources is required. Potential conflicts include the following:

- Two management stations want to use simultaneously Resources that together would exceed the capability of the device.

- A management station uses a significant amount of resources for a long period of time.

- A management station uses resources and then crashes, forgetting to free the resources so others may use them.

A mechanism is provided for each management station initiated function in this MIB to avoid these conflicts and to help resolve them when they occur. Each function has a label identifying the initiator (owner) of the function. This label is set by the initiator to provide for the following possibilities:

- A management station may recognize resources it owns and no longer needs.

- A network operator can find the management station that owns the resource and negotiate for it to be freed.

- A network operator may decide to unilaterally free resources another network operator has reserved.

- Upon initialization, a management station may recognize resources it had reserved in the past. With this information it may free the resources if it no longer needs them.

Management stations and probes should support any format of the owner string dictated by the local policy of the organization. It is suggested that this name contain one or more of the following:

- IP address

- Management station name

- Network manager's name

- Location

- Phone number

This information will help users share the resources more effectively.

There is often default functionality that the device wants to set up. The resources associated with this functionality are then owned by the device itself. In this case, the device sets the relevant owner object to a string starting with "monitor". Indiscriminate modification of the monitor-owned configuration by network management stations is discouraged. In fact, a network management station should only modify these objects under the direction of the administrator of the probe, often the network administrator.

When a network management station wants to use a function in a monitor, it is encouraged to first scan the control table of that function to find an instance with similar parameters to share. This is especially true for those instances owned by the monitor, that can be assumed to change infrequently. If a management station decides to share an instance owned by another management station, it should understand that the management station that owns the instance may indiscriminately modify or delete it.

5.4.5 Row Addition Among Multiple Management Stations

The addition of new rows is achieved using the method described in [9]. In this MIB, rows are often added to a table in order to configure a function. This configuration usually involves parameters that control the operation of the function. The agent must check these parameters to make sure they are appropriate given the restrictions defined in this MIB; as well as any implementation-specific restrictions such as lack of resources. The agent implementor may be confused as to when to check these parameters and when to signal to the management station that the parameters are invalid. There are two opportunities:

• When the management station sets each parameter object

• When the management station sets the entry status object to valid

If the latter is chosen, it would be unclear to the management station which of the several parameters was invalid and caused the badValue error to be emitted. Thus, wherever possible, the implementor should choose the former as it will provide more information to the management station.

A problem can arise when multiple management stations attempt to set configuration information simultaneously using SNMP. When this involves the addition of a new conceptual row in the same control table, the managers may collide, attempting to create the same entry. To guard against these collisions, each such control entry contains a status object with special semantics that help to arbitrate among the managers. If an attempt is made with the row addition mechanism to create such a status object and that object already exists, an error is returned. When more than one manager simultaneously attempts to create the same conceptual row, only the first will succeed. The others will receive an error.

6
Statistics Group

Implementation of the Statistics group is optional.

The Statistics group contains statistics measured by the probe for each monitored interface on this device. These statistics take the form of free-running counters that start from zero when a valid entry is created.

This group currently has statistics defined for the Ethernet interface. Each etherStatsEntry contains statistics for one interface. The probe must create one entry for each monitored interface on the device.

etherStatsTable

{ statistics }

A list of Ethernet statistics entries.

etherStatsEntry

{ etherStatsTable 1 }

A collection of statistics kept for a particular Ethernet interface:

Entry	Description
etherStatsIndex	INTEGER (1 . . . 65535)
etherStatsDataSource	Object Identifier
etherStatsDropEvents	Counter
etherStatsOctets	Counter
etherStatsPkts	Counter
etherStatsBroadcastPkts	Counter
etherStatsMulticastPkts	Counter
etherStatsCRCAlignErrors	Counter
etherStatsUndersizePkts	Counter
etherStatsOversizePkts	Counter
etherStatsFragments	Counter
etherStatsJabbers	Counter
etherStatsCollisions	Counter
etherStatsPkts64Octets	Counter
etherStatsPkts65to127Octets	Counter
etherStatsPkts128to255Octets	Counter
etherStatsPkts256to511Octets	Counter
etherStatsPkts512to1023Octets	Counter
etherStatsPkts1024to1518Octets	Counter
etherStatsOwner	OwnerString
etherStatsStatus	INTEGER

etherStatsIndex

{ etherStatsEntry 1 }

The value of this object uniquely identifies this etherStats entry.

etherStatsDataSource

{ etherStatsEntry 2 }

This object identifies the source of the data it is configured to analyze. This source can be any Ethernet interface on this device. To identify a particular interface, this object shall identify the instance of the ifIndex object, defined in [4,6], for the desired interface.

For example, if an entry were to receive data from interface 1, this object would be set to ifIndex.1.

The statistics in this group reflect all packets on the local network segment attached to the identified interface.

This object may not be modified if the associated etherStatsStatus object is equal to valid(1).

etherStatsDropEvents

{ etherStatsEntry 3 }

The total number of events where packets were dropped by the probe due to lack of resources. Note that this number is not the number of packets dropped. It is the number of times this condition has been detected.

etherStatsOctets

{ etherStatsEntry 4 }

The total number of octets of data (including those in bad packets) received on the network (excluding framing bits but including FCS octets).

etherStatsPkts

{ etherStatsEntry 5 }

The total number of packets (including error packets) received.

etherStatsBroadcastPkts

{ etherStatsEntry 6 }

The total number of good packets received that were directed to the broadcast address.

etherStatsMulticastPkts

{ etherStatsEntry 7 }

The total number of good packets received that were directed to a multicast address. Note that this number does not include packets directed to the broadcast address.

etherStatsCRCAlignErrors

{ etherStatsEntry 8 }

The total number of packets received that had a length (excluding framing bits but including FCS octets) of between 64 and 1518 octets, inclusive, but were not an integral number of octets in length or had a bad Frame Check Sequence (FCS).

etherStatsUndersizePkts

{ etherStatsEntry 9 }

The total number of packets received that were less than 64 octets long (excluding framing bits but including FCS octets) and were otherwise well formed.

etherStatsOversizePkts

{ etherStatsEntry 10 }

The total number of packets received that were longer than 1518 octets (excluding framing bits but including FCS octets) and were otherwise well formed.

etherStatsFragments

{ etherStatsEntry 11 }

The total number of packets received that were not an integral number of octets in length or that had a bad Frame Check Sequence (FCS), and were less than 64 octets in length (excluding framing bits but including FCS octets).

etherStatsJabbers

{ etherStatsEntry 12 }

The total number of packets received that were longer than 1518 octets (excluding framing bits but including FCS octets), and were not an integral number of octets in length or had a bad Frame Check Sequence (FCS).

etherStatsCollisions

{ etherStatsEntry 13 }

The best estimate of the total number of collisions on this Ethernet segment.

etherStatsPkts64Octets

{ etherStatsEntry 14 }

The total number of packets (including error packets) received that were 64 octets in length (excluding framing bits but including FCS octets).

etherStatsPkts65to127Octets

{ etherStatsEntry 15 }

The total number of packets (including error packets) received that were between 65 and 127 octets in length inclusive (excluding framing bits but including FCS octets).

etherStatsPkts128to255Octets

{ etherStatsEntry 16 }

The total number of packets (including error packets) received that were between 128 and 255 octets in length inclusive (excluding framing bits but including FCS octets).

etherStatsPkts256to511Octets

{ etherStatsEntry 17 }

The total number of packets (including error packets) received that were between 256 and 511 octets in length inclusive (excluding framing bits but including FCS octets).

etherStatsPkts512to1023Octets

{ etherStatsEntry 18 }

The total number of packets (including error packets) received that were between 512 and 1023 octets in length inclusive (excluding framing bits but including FCS octets).

etherStatsPkts1024to1518Octets

{ etherStatsEntry 19 }

The total number of packets (including error packets) received that were between 1024 and 1518 octets in length inclusive (excluding framing bits but including FCS octets).

etherStatsOwner

{ etherStatsEntry 20 }

The entity that configured this entry and is using the resources assigned to it.

etherStatsStatus

{ etherStatsEntry 21 }

The status of this etherStats entry.

7
History Group

Implementation of the History group is optional.

The History group records periodic statistical samples from a network and stores them for later retrieval. The historyControl table stores configuration entries that each define an interface, polling period, and other parameters. Once samples are taken, their data is stored in an entry in a media-specific table. Each such entry defines one sample, and is associated with the historyControlEntry that caused the sample to be taken. The media-specific table for Ethernet networks is defined as etherHistoryTable.

If the probe keeps track of the time of day, it should start the first sample of the history at such a time that when the next hour of the day begins, a sample is started at that instant. This tends to make more user-friendly reports, and enables comparison of reports from different probes that have relatively accurate time of day.

The monitor is encouraged to add two history control entries per monitored interface upon initialization that describe a short term and a long term polling period. Suggested parameters are 30 seconds for the short term polling period and 30 minutes for the long term polling period.

HistoryControl

historyControlTable
{ history 1 }

A list of history control entries.

historyControlEntry
{ historyControlTable 1 }

A list of parameters that set up a periodic sampling of statistics:

Entry	Description
historyControlIndex	INTEGER (1 . . . 65535)
historyControlDataSource	OBJECT IDENTIFIER
historyControlBucketsRequested	INTEGER (1 . . . 65535)
historyControlBucketsGranted	INTEGER (1 . . . 65535)
historyControlInterval	INTEGER (1 . . . 3600)
historyControlOwner	OwnerString
historyControlStatus	INTEGER

historyControlIndex
{ historyControlEntry 1 }

An index that uniquely identifies an entry in the historyControl table. Each such entry defines a set of samples at a particular interval for an interface on the device.

historyControlDataSource
{ historyControlEntry 2 }

This object identifies the source of the data for which historical data was collected and placed in a media-specific table on behalf of this historyControlEntry. This source can be any interface on this device. To identify a particular interface, this object shall identify the instance of the ifIndex object for the desired interface. For example, if an entry were to receive data from interface 1, this object would be set to ifIndex.1.

The statistics in this group reflect all packets on the local network segment attached to the identified interface.

This object may not be modified if the associated historyControlStatus object is equal to valid(1).

historyControlBucketsRequested

{ historyControlEntry 3 }

The requested number of discrete time intervals where data is to be saved in the part of the media-specific table associated with this historyControl entry.

When this object is created or modified, the probe should set historyControl-BucketsGranted as closely to this object as is possible for the particular probe implementation and available resources.

The default value for historyControlBuckets is 50.

historyControlBucketsGranted

{ historyControlEntry 4 }

The number of discrete sampling intervals where data shall be saved in the part of the media-specific table associated with this historyControl entry.

When the associated historyControlBucketsRequested object is created or modified, the probe should set this object as closely to the requested value as is possible for the particular probe implementation and available resources. The probe must not lower this value except as a result of a modification to the associated historyControlBucketsRequested object.

There will be times when the actual number of buckets associated with this entry is less than the value of this object. In this case, at the end of each sampling interval, a new bucket will be added to the media-specific table.

When the number of buckets reaches the value of this object and a new bucket is to be added to the media-specific table, the oldest bucket associated with this historyControlEntry shall be deleted by the agent so that the new bucket can be added.

When the value of this object changes to a value less than the current value, entries are deleted from the media-specific table associated with this historyControlEntry. Enough of the oldest of these entries shall be deleted by the agent so that their number remains less than or equal to the new value of this object.

When the value of this object changes to a value greater than the current value, the number of associated media-specific entries may be allowed to grow.

historyControlInterval

{ historyControlEntry 5 }

The interval in seconds where the data is sampled for each bucket in the part of the media-specific table associated with this historyControl entry. This interval can be set to any number of seconds between 1 and 3600 (1 hour).

Because the counters in a bucket may overflow at their maximum value with no indication, a prudent manager will take into account the possibility of overflow in any of the associated counters. It is important to consider the minimum time in which any counter could overflow on a particular media type and set the historyControlInterval object to a value less than this interval. This is typically most important for the "octets" counter in any media-specific table. For example, on an Ethernet network, the etherHistoryOctets counter could overflow in about one hour at the Ethernet's maximum utilization.

This object may not be modified if the associated historyControlStatus object is equal to valid(1).

The default value for historyControlInterval is 1800 (30 minutes).

historyControlOwner

{ historyControlEntry 6 }

The entity that configured this entry and is therefore using the resources assigned to it.

historyControlStatus

{ historyControlEntry 7 }

The status of this historyControl entry. Each instance of the media-specific table associated with this historyControlEntry will be deleted by the agent if this historyControlEntry is not equal to valid(1).

EtherHistory

etherHistoryTable
{ history 2 }

A list of Ethernet history entries.

etherHistoryEntry
{ etherHistoryTable 1 }

An historical sample of Ethernet statistics on a particular Ethernet interface. This sample is associated with the historyControlEntry that set up the parameters for a regular collection of these samples:

Entry	Description
etherHistoryIndex	INTEGER (1 . . . 65535)
etherHistorySampleIndex	INTEGER
etherHistoryIntervalStart	TimeTicks
etherHistoryDropEvents	Counter
etherHistoryOctets	Counter
etherHistoryPkts	Counter
etherHistoryBroadcastPkts	Counter
etherHistoryMulticastPkts	Counter
etherHistoryCRCAlignErrors	Counter
etherHistoryUndersizePkts	Counter
etherHistoryOversizePkts	Counter
etherHistoryFragments	Counter
etherHistoryJabbers	Counter
etherHistoryCollisions	Counter
etherHistoryUtilization	INTEGER (0 . . . 10000)

etherHistoryIndex
{ etherHistoryEntry 1 }

The history of which this entry is a part. The history identified by a particular value of this index is the same history as identified by the same value of historyControlIndex.

etherHistorySampleIndex

{ etherHistoryEntry 2 }

An index that uniquely identifies the particular sample this entry represents among all samples associated with the same historyControlEntry. This index starts at 1 and increases by one as each new sample is taken.

etherHistoryIntervalStart

{ etherHistoryEntry 3 }

The value of sysUpTime at the start of the interval where this sample was measured. If the probe keeps track of the time of day, it should start the first sample of the history at such a time that when the next hour of the day begins, a sample is started at that instant. Note that following this rule may require the probe to delay collecting the first sample of the history, as each sample must be of the same interval. Also note that the sample that is currently being collected is not accessible in this table until the end of its interval.

etherHistoryDropEvents

{ etherHistoryEntry 4 }

The total number of events where packets were dropped by the probe due to lack of resources during this interval. Note that this number is not the number of packets dropped. It is the number of times this condition has been detected.

etherHistoryOctets

{ etherHistoryEntry 5 }

The total number of octets of data (including those in bad packets) received on the network (excluding framing bits but including FCS octets).

etherHistoryPkts

{ etherHistoryEntry 6 }

The number of packets (including error packets) received during this sampling interval.

etherHistoryBroadcastPkts

{ etherHistoryEntry 7 }

The number of good packets received during this sampling interval that were directed to the broadcast address.

etherHistoryMulticastPkts

{ etherHistoryEntry 8 }

The number of good packets received during this sampling interval that were directed to a multicast address. Note that this number does not include packets addressed to the broadcast address.

etherHistoryCRCAlignErrors

{ etherHistoryEntry 9 }

The number of packets received during this sampling interval that had a length (excluding framing bits but including FCS octets) between 64 and 1518 octets, inclusive, but were not an integral number of octets in length or had a bad Frame Check Sequence (FCS).

etherHistoryUndersizePkts

{ etherHistoryEntry 10 }

The number of packets received during this interval that were less than 64 octets long (excluding framing bits but including FCS octets) and were otherwise well formed.

etherHistoryOversizePkts

{ etherHistoryEntry 11 }

The number of packets received during this interval that were longer than 1518 octets (excluding framing bits but including FCS octets) but were otherwise well formed.

etherHistoryFragments

{ etherHistoryEntry 12 }

The total number of packets received during this sampling interval that were not an integral number of octets in length or that had a bad Frame Check Sequence (FCS), and were less than 64 octets in length (excluding framing bits but including FCS octets).

etherHistoryJabbers

{ etherHistoryEntry 13 }

The number of packets received during this interval that were longer than 1518 octets (excluding framing bits but including FCS octets), and were not an integral number of octets in length or had a bad Frame Check Sequence (FCS).

etherHistoryCollisions

{ etherHistoryEntry 14 }

The best estimate of the total number of collisions on this Ethernet segment during this interval.

etherHistoryUtilization

{ etherHistoryEntry 15 }

The best estimate of the mean physical layer network utilization on this interface during this interval, in hundredths of a percent.

8

Alarm Group

Implementation of the Alarm group is optional.

The Alarm group requires the implementation of the Event group.

The Alarm group periodically takes statistical samples from variables in the probe and compares them to configured thresholds. The alarm table stores configuration entries that define each of the following parameters:

- Variable
- Polling period
- Threshold

If a sample is found to cross the threshold values, an event is generated. Only variables that resolve to an ASN.1 primitive type of INTEGER (INTEGER, Counter, Gauge, or TimeTicks) may be monitored in this way.

This function has a hysteresis mechanism to limit the generation of events. This mechanism generates one event as a threshold is crossed in the appropriate direction. No more events are generated for that threshold until the opposite threshold is crossed.

In the case of sampling a deltaValue, a probe may implement this mechanism with more precision if it takes a delta sample twice per period. Each time it takes a sample, it compares the sum of the latest two samples to the threshold. This allows the detection of threshold crossings that span the sampling boundary. Note that this does not require any special configuration of the threshold value. It is suggested that probes implement this more precise algorithm.

alarmTable

{ alarm 1 }

A list of alarm entries.

alarmEntry

{ alarmTable 1 }

A list of parameters that set up a periodic checking for alarm conditions:

Entry	Description
alarmIndex	INTEGER (1 . . . 65535)
alarmInterval	INTEGER
alarmVariable	OBJECT IDENTIFIER
alarmSampleType	INTEGER
alarmValue	INTEGER
alarmStartupAlarm	INTEGER
alarmRisingThreshold	INTEGER
alarmFallingThreshold	INTEGER
alarmRisingEventIndex	INTEGER (1 . . . 65535)
alarmFallingEventIndex	INTEGER (1 . . . 65535)
alarmOwner	OwnerString
alarmStatus	INTEGER

alarmIndex

{ alarmEntry 1 }

An index that uniquely identifies an entry in the alarm table. Each such entry defines a diagnostic sample at a particular interval for an object on the device.

alarmInterval

{ alarmEntry 2 }

The interval in seconds where the data is sampled and compared with the rising and falling thresholds. When setting this variable, care should be given to ensure that the variable being monitored will not exceed $2^{31} - 1$ and roll over the alarmValue object during the interval.

This object may not be modified if the associated alarmStatus object is equal to valid(1).

alarmVariable

{ alarmEntry 3 }

The object identifier of the particular variable to be sampled. Only variables that resolve to an ASN.1 primitive type of INTEGER (INTEGER, Counter, Gauge, or TimeTicks) may be sampled. SNMP access control is articulated entirely in terms of the contents of MIB views. No access control mechanism exists that can restrict the value of this object to identify only those objects that exist in a particular MIB view. There is thus no acceptable means of restricting the read access that could be obtained through the alarm mechanism. The probe must only grant write access to this object in those views that have read access to all objects on the probe.

During a set operation, if the supplied variable name is not available in the selected MIB view, a badValue error must be returned. If at any time the variable name of an established alarmEntry is no longer available in the selected MIB view, the probe must change the status of this alarmEntry to invalid(4).

This object may not be modified if the associated alarmStatus object is equal to valid(1).

alarmSampleType

{ alarmEntry 4 }

The method of sampling the selected variable and calculating the value to be compared against the thresholds. If the value of this object is absoluteValue(1), the value of the selected variable will be compared directly with the thresholds at the end of the sampling interval. If the value of this object is deltaValue(2), the value of the selected variable at the last sample will be subtracted from the current value, and the difference compared with the thresholds.

This object may not be modified if the associated alarmStatus object is equal to valid(1).

alarmValue

{ alarmEntry 5 }

The value of the statistic during the last sampling period. The value during the current sampling period is not made available until the period is completed.

alarmStartupAlarm

{ alarmEntry 6 }

The alarm that may be sent when this entry is first set to valid.

If . . .	And . . .	Then . . .
The first sample after this entry becomes valid is greater than or equal to the risingThreshold	AlarmStartupAlarm is equal to risingAlarm(1) or risingOrFallingAlarm(3)	A single rising alarm will be generated.
The first sample after this entry becomes valid is less than or equal to the fallingThreshold	AlarmStartupAlarm is equal to fallingAlarm(2) or risingOrFallingAlarm(3)	A single falling alarm will be generated.

This object may not be modified if the associated alarmStatus object is equal to valid(1).

alarmRisingThreshold

{ alarmEntry 7 }

A threshold for the sampled statistic.

If . . .	And . . .	Then . . .
The current sampled value is greater than or equal to this threshold	The value at the last sampling interval was less than this threshold	A single event will be generated.
The first sample after this entry becomes valid is greater than or equal to this threshold	The associated alarm-StartupAlarm is equal to risingAlarm(1) or risingOrFallingAlarm(3)	A single event will also be generated.

After a rising event is generated, another such event will not be generated until the sampled value falls below this threshold and reaches the alarmFallingThreshold.

This object may not be modified if the associated alarmStatus object is equal to valid(1).

alarmFallingThreshold

{ alarmEntry 8 }

A threshold for the sampled statistic.

If . . .	And . . .	Then . . .
The current sampled value is less than or equal to this threshold	The value at the last sampling interval was greater than this threshold	A single event will be generated.
The first sample after this entry becomes valid is less than or equal to this threshold	The associated alarmStartupAlarm is equal to fallingAlarm(2) or risingOrFallingAlarm(3)	A single event will also be generated.

After a falling event is generated, another such event will not be generated until the sampled value rises above this threshold and reaches the alarmRisingThreshold.

This object may not be modified if the associated alarmStatus object is equal to valid(1).

alarmRisingEventIndex

{ alarmEntry 9 }

The index of the eventEntry that is used when a rising threshold is crossed. The eventEntry identified by a particular value of this index is the same as identified by the same value of the eventIndex object. If there is no corresponding entry in the eventTable, then no association exists. In particular, if this value is zero, no associated event will be generated, as zero is not a valid event index.

This object may not be modified if the associated alarmStatus object is equal to valid(1).

alarmFallingEventIndex

{ alarmEntry 10 }

The index of the eventEntry that is used when a falling threshold is crossed. The eventEntry identified by a particular value of this index is the same as identified by the same value of the eventIndex object. If there is no corresponding entry in the eventTable, then no association exists. In particular, if this value is zero, no associated event will be generated, as zero is not a valid event index.

This object may not be modified if the associated alarmStatus object is equal to valid(1).

alarmOwner

{ alarmEntry 11 }

The entity that configured this entry and is using the resources assigned to it.

alarmStatus

{ alarmEntry 12 }

The status of this alarm entry.

9
Host Group

Implementation of the Host group is optional.

The Host group discovers new hosts on the network by keeping a list of source and destination MAC addresses seen in good packets. For each of these addresses, the Host group keeps a set of statistics. The hostControlTable controls which interfaces this function is performed on, and contains some information about the process. On behalf of each hostControlEntry, data is collected on an interface and placed in both the hostTable and the hostTimeTable. If the monitoring device finds itself short of resources, it may delete entries as needed. It is suggested that the device delete the least recently used entries first. The hostTable contains entries for each address discovered on a particular interface. Each entry contains statistical data about that host. This table is indexed by the MAC address of the host, through which a random access may be achieved.

The hostTimeTable contains data in the same format as the hostTable, and must contain the same set of hosts, but is indexed using hostTimeCreationOrder rather than hostAddress. The hostTimeCreationOrder is an integer that reflects the relative order in which a particular entry was discovered and inserted into the table. As this order (index) is among those entries currently in the table, the index for a particular entry may change if an (earlier) entry is deleted. Thus, the association between hostTimeCreationOrder and hostTimeEntry may be broken at any time.

The hostTimeTable has two important uses. The first is the fast download of this potentially large table. Because the index of this table runs from 1 to the size of the table, inclusive, its values are predictable. This allows very efficient packing of variables into SNMP PDUs and allows a table transfer to have multiple packets outstanding. These benefits increase transfer rates tremendously.

The second use of the hostTimeTable is the efficient discovery by the management station of new entries added to the table. After the management station has downloaded the entire table, it knows that new entries will be added immediately after the end of the current table. It can thus detect new entries there and retrieve them easily.

Because the association between hostTimeCreationOrder and hostTimeEntry may be broken at any time, the management station must monitor the related hostControlLastDeleteTime object. When the management station thus detects a deletion, it must assume that any such associations have been broken, and invalidate any it has stored locally. This includes restarting any download of the hostTimeTable that may have been in progress, as well as rediscovering the end of the hostTimeTable so that it may detect new entries. If the management station does not detect the broken association, it may continue to refer to a particular host by its creationOrder while unwittingly retrieving the data associated with another host entirely. If this happens while downloading the host table, the management station may fail to download all of the entries in the table.

hostControlTable
{ hosts 1 }

A list of host table control entries.

hostControlEntry
{ hostControlTable 1 }

A list of parameters that set up the discovery of hosts on a particular interface and the collection of statistics about these hosts:

Entry	Description
hostControlIndex	INTEGER (1 . . . 65535)
hostControlDataSource	OBJECT IDENTIFIER
hostControlTableSize	INTEGER
hostControlLastDeleteTime	TimeTicks
hostControlOwner	OwnerString
hostControlStatus	INTEGER

hostControlIndex
{ hostControlEntry 1 }

An index that uniquely identifies an entry in the hostControl table. Each entry defines a function that discovers hosts on a particular interface and places statistics about them in the hostTable and the hostTimeTable on behalf of this hostControlEntry.

hostControlDataSource

{ hostControlEntry 2 }

This object identifies the source of the data for this instance of the host function. This source can be any interface on this device. To identify a particular interface, this object shall identify the instance of the ifIndex object for the desired interface. For example, if an entry were to receive data from interface 1, this object would be set to ifIndex.1.

The statistics in this group reflect all packets on the local network segment attached to the identified interface.

This object may not be modified if the associated hostControlStatus object is equal to valid(1).

hostControlTableSize

{ hostControlEntry 3 }

The number of hostEntries in the hostTable and the hostTimeTable associated with this hostControlEntry.

hostControlLastDeleteTime

{ hostControlEntry 4 }

The value of sysUpTime when the last entry was deleted from the portion of the hostTable associated with this hostControlEntry. If no deletions have occurred, this value shall be zero.

hostControlOwner

{ hostControlEntry 5 }

The entity that configured this entry and is using the resources assigned to it.

hostControlStatus

{ hostControlEntry 6 }

The status of this hostControl entry.

If this object is not equal to valid(1), all associated entries in the hostTable, hostTimeTable, and the hostTopNTable shall be deleted by the agent.

hostTable

{ hosts 2 }

A list of host entries.

hostEntry

{ hostTable 1 }

A collection of statistics for a particular host that has been discovered on an interface of this device:

Entry	Description
hostAddress	OCTET STRING
hostCreationOrder	INTEGER (1 . . . 65535)
hostIndex	INTEGER (1 . . . 65535)
hostInPkts	Counter
hostOutPkts	Counter
hostInOctets	Counter
hostOutOctets	Counter
hostOutErrors	Counter
hostOutBroadcastPkts	Counter
hostOutMulticastPkts	Counter

hostAddress

{ hostEntry 1 }

The physical address of this host.

hostCreationOrder

{ hostEntry 2 }

An index that defines the relative ordering of the creation time of hosts captured for a particular hostControlEntry. This index shall be between 1 and n, where n is the value of the associated hostControlTableSize.

The ordering of the indexes is based on the order of each entry's insertion into the table, where entries added earlier have a lower index value than entries added later.

It is important to note that the order for a particular entry may change as an (earlier) entry is deleted from the table. Because this order may change, management stations should make use of the hostControlLastDeleteTime variable in the hostControlEntry associated with the relevant portion of the hostTable. By observing this variable, the management station may detect the circumstances where a previous association between a value of hostCreationOrder and a hostEntry may no longer hold.

hostIndex

{ hostEntry 3 }

The set of collected host statistics of which this entry is a part. The set of hosts identified by a particular value of this index is associated with the hostControlEntry as identified by the same value of hostControlIndex.

hostInPkts

{ hostEntry 4 }

The number of packets without errors transmitted to this address since it was added to the hostTable.

hostOutPkts

{ hostEntry 5 }

The number of packets including errors transmitted by this address since it was added to the hostTable.

hostInOctets

{ hostEntry 6 }

The number of octets transmitted to this address since it was added to the hostTable (excluding framing bits but including FCS octets), except for those octets in packets that contained errors.

hostOutOctets

{ hostEntry 7 }

The number of octets transmitted by this address since it was added to the hostTable (excluding framing bits but including FCS octets), including those octets in packets that contained errors.

hostOutErrors

{ hostEntry 8 }

The number of error packets transmitted by this address since this host was added to the hostTable.

hostOutBroadcastPkts

{ hostEntry 9 }

The number of good packets transmitted by this address that were directed to the broadcast address since this host was added to the hostTable.

hostOutMulticastPkts

{ hostEntry 10 }

The number of good packets transmitted by this address that were directed to a multicast address since this host was added to the hostTable. Note that this number does not include packets directed to the broadcast address.

hostTimeTable

{ hosts 3 }

A list of time-ordered host table entries.

hostTimeEntry

{ hostTimeTable 1 }

A collection of statistics for a particular host that has been discovered on an interface of this device. This collection includes the relative ordering of the creation time of this object:

Entry	Description
hostTimeAddress	OCTET STRING
hostTimeCreationOrder	INTEGER (1 . . . 65535)
hostTimeIndex	INTEGER (1 . . . 65535)
hostTimeInPkts	Counter
hostTimeOutPkts	Counter
hostTimeInOctets	Counter
hostTimeOutOctets	Counter
hostTimeOutErrors	Counter
hostTimeOutBroadcastPkts	Counter
hostTimeOutMulticastPkts	Counter

hostTimeAddress

{ hostTimeEntry 1 }

The physical address of this host.

hostTimeCreationOrder

{ hostTimeEntry 2 }

An index that uniquely identifies an entry in the hostTime table among those entries associated with the same hostControlEntry. This index shall be between 1 and *n*, where *n* is the value of the associated hostControlTableSize.

The ordering of the indexes is based on the order of each entry's insertion into the table, where entries added earlier have a lower index value than entries added later. Thus, the management station has the ability to learn of new entries added to this table without downloading the entire table.

It is important to note that the index for a particular entry may change as an (earlier) entry is deleted from the table. Because this order may change, management stations should make use of the hostControlLastDeleteTime variable in the hostControlEntry associated with the relevant portion of the hostTimeTable. By observing this variable, the management station may detect the circumstances where a download of the table may have missed entries, and where a previous association between a value of hostTimeCreationOrder and a hostTimeEntry may no longer hold.

hostTimeIndex

{ hostTimeEntry 3 }

The set of collected host statistics of which this entry is a part. The set of hosts identified by a particular value of this index is associated with the hostControlEntry as identified by the same value of hostControlIndex.

hostTimeInPkts

{ hostTimeEntry 4 }

The number of packets without errors transmitted to this address since it was added to the hostTimeTable.

hostTimeOutPkts

{ hostTimeEntry 5 }

The number of packets including errors transmitted by this address since it was added to the hostTimeTable.

hostTimeInOctets

{ hostTimeEntry 6 }

The number of octets transmitted to this address since it was added to the hostTimeTable (excluding framing bits but including FCS octets), except for those octets in packets that contained errors.

hostTimeOutOctets

{ hostTimeEntry 7 }

The number of octets transmitted by this address since it was added to the hostTimeTable (excluding framing bits but including FCS octets), including those octets in packets that contained errors.

hostTimeOutErrors

{ hostTimeEntry 8 }

The number of error packets transmitted by this address since this host was added to the hostTimeTable.

hostTimeOutBroadcastPkts

{ hostTimeEntry 9 }

The number of good packets transmitted by this address that were directed to the broadcast address since this host was added to the hostTimeTable.

hostTimeOutMulticastPkts

{ hostTimeEntry 10 }

The number of good packets transmitted by this address that were directed to a multicast address since this host was added to the hostTimeTable. Note that this number does not include packets directed to the broadcast address.

10
Host Top N Group

Implementation of the Host Top N group is optional.

The Host Top N group requires the implementation of the Host group.

The Host Top N group is used to prepare reports that describe the hosts that are at the top of a list that has been ordered by one of their statistics. The available statistics are samples of one of their base statistics, over an interval specified by the management station. Thus, these statistics are rate based. The management station also selects how many such hosts are reported.

The hostTopNControlTable is used to initiate the generation of such a report. The management station may select the parameters of such a report. For example:

- Which interface
- Which statistic
- How many hosts
- The start time of the sampling
- The stop time of the sampling

When the report is prepared, entries are created in the hostTopNTable associated with the relevant hostTopNControlEntry. These entries are static for each report after it has been prepared.

hostTopNControlTable

{ hostTopN 1 }

A list of top N host control entries.

hostTopNControlEntry

{ hostTopNControlTable 1 }

A set of parameters that control the creation of a report of the top N hosts according to several metrics:

Entry	Description
hostTopNControlIndex	INTEGER (1 . . . 65535)
hostTopNHostIndex	INTEGER (1 . . . 65535)
hostTopNRateBase	INTEGER
hostTopNTimeRemaining	INTEGER
hostTopNDuration	INTEGER
hostTopNRequestedSize	INTEGER
hostTopNGrantedSize	INTEGER
hostTopNStartTime	TimeTicks
hostTopNOwner	OwnerString
hostTopNStatus	INTEGER

hostTopNControlIndex

{ hostTopNControlEntry 1 }

An index that uniquely identifies an entry in the hostTopNControl table. Each entry defines one top N report prepared for one interface.

hostTopNHostIndex

{ hostTopNControlEntry 2 }

The host table where a top N report will be prepared on behalf of this entry. The host table identified by a particular value of this index is associated with the same host table as identified by the same value of hostIndex.

This object may not be modified if the associated hostTopNStatus object is equal to valid(1).

hostTopNRateBase

{ hostTopNControlEntry 3 }

> hostTopNInPkts(1)
> hostTopNOutPkts(2)
> hostTopNInOctets(3)
> hostTopNOutOctets(4)
> hostTopNOutErrors(5)
> hostTopNOutBroadcastPkts(6)
> hostTopNOutMulticastPkts(7)

The variable for each host on which the hostTopNRate variable is based.

This object may not be modified if the associated hostTopNStatus object is equal to valid(1).

hostTopNTimeRemaining

{ hostTopNControlEntry 4 }

The number of seconds left in the report currently being collected. When this object is modified by the management station, a new collection is started, possibly aborting a currently running report. The new value is used as the requested duration of this report, which is loaded into the associated hostTopNDuration object.

When this object is set to a nonzero value, any associated hostTopNEntries shall be made inaccessible by the monitor. While the value of this object is nonzero, it decrements by one per second until it reaches zero. During this time, all associated hostTopNEntries shall remain inaccessible. At the time that this object decrements to zero, the report is made accessible in the hostTopNTable. Thus, the hostTopNTable needs to be created only at the end of the collection interval.

The default value of hostTopNTimeRemaining is 0 (zero).

hostTopNDuration

{ hostTopNControlEntry 5 }

The number of seconds this report has collected during the last sampling interval. If this report is currently being collected, the number of seconds this report is being collected during this sampling interval.

When the associated hostTopNTimeRemaining object is set, this object shall be set by the probe to the same value and shall not be modified until the next time the hostTopNTimeRemaining is set.

This value shall be zero if no reports have been requested for this hostTopNControlEntry. The default value of hostTopNDuration is 0 (zero).

hostTopNRequestedSize

{ hostTopNControlEntry 6 }

The maximum number of hosts requested for the top N table.

When this object is created or modified, the probe should set hostTopNGranted-Size as closely to this object as is possible for the particular probe implementation and available resources.

The default value for hostTopNRequestedSize is 10.

hostTopNGrantedSize

{ hostTopNControlEntry 7 }

The maximum number of hosts in the top N table.

When the associated hostTopNRequestedSize object is created or modified, the probe should set this object as closely to the requested value as is possible for the particular implementation and available resources. The probe must not lower this value except as a result of a set to the associated hostTopNRequestedSize object.

Hosts with the highest value of hostTopNRate shall be placed in this table in decreasing order of this rate until there is no more room or until there are no more hosts.

hostTopNStartTime

{ hostTopNControlEntry 8 }

The value of sysUpTime when this top N report was last started. In other words, this is the time that the associated hostTopNTimeRemaining object was modified to start the requested report.

hostTopNOwner

{ hostTopNControlEntry 9 }

The entity that configured this entry and is using the resources assigned to it.

hostTopNStatus

{ hostTopNControlEntry 10 }

The status of this hostTopNControl entry.

If this object is not equal to valid(1), all associated hostTopNEntries shall be deleted by the agent.

hostTopNTable

{ hostTopN 2 }

A list of top N host entries.

hostTopNEntry

{ hostTopNTable 1 }

A set of statistics for a host that is part of a top N report:

Entry	Description
hostTopNReport	INTEGER (1 . . . 65535)
hostTopNIndex	INTEGER (1 . . . 65535)
hostTopNAddress	OCTET STRING
hostTopNRate	INTEGER

hostTopNReport

{ hostTopNEntry 1 }

This object identifies the top N report of which this entry is a part. The set of hosts identified by a particular value of this object is part of the same report as identified by the same value of the hostTopNControlIndex object.

hostTopNIndex

{ hostTopNEntry 2 }

An index that uniquely identifies an entry in the hostTopNTable among those in the same report. This index is between 1 and *n*, where *n* is the number of entries in this table.

Increasing values of hostTopNIndex shall be assigned to entries with decreasing values of hostTopNRate until index N is assigned to the entry with the lowest value of hostTopNRate or there are no more hostTopNEntries.

hostTopNAddress

{ hostTopNEntry 3 }

The physical address of this host.

hostTopNRate

{ hostTopNEntry 4 }

The amount of change in the selected variable during this sampling interval. The selected variable is this host's instance of the object selected by hostTopNRate-Base.

11
Matrix Group

Implementation of the Matrix group is optional.

The Matrix group consists of the matrixControlTable, matrixSDTable, and the matrixDSTable. These tables store statistics for a particular conversation between two addresses. As the device detects a new conversation, including those to a non-unicast address, it creates a new entry in both of the matrix tables. It must only create new entries based on information received in good packets. If the monitoring device finds itself short of resources, it may delete entries as needed. It is suggested that the device delete the least recently used entries first.

matrixControlTable
{ matrix 1 }

A list of information entries for the traffic matrix on each interface.

matrixControlEntry
{ matrixControlTable 1 }

Information about a traffic matrix on a particular interface:

Entry	Description
matrixControlIndex	INTEGER (1 . . . 65535)
matrixControlDataSource	OBJECT IDENTIFIER
matrixControlTableSize	INTEGER
matrixControlLastDeleteTime	TimeTick
matrixControlOwner	OwnerString
matrixControlStatus	INTEGER

matrixControlIndex
{ matrixControlEntry 1 }

An index that uniquely identifies an entry in the matrixControl table. Each such entry defines a function that discovers conversations on a particular interface and places statistics about them in the matrixSDTable and the matrixDSTable on behalf of this matrixControlEntry.

matrixControlDataSource
{ matrixControlEntry 2 }

This object identifies the source of the data from which this entry creates a traffic matrix. This source can be any interface on this device. To identify a particular interface, this object shall identify the instance of the ifIndex object for the desired interface. For example, if an entry were to receive data from interface 1, this object would be set to ifIndex.1.

The statistics in this group reflect all packets on the local network segment attached to the identified interface.

This object may not be modified if the associated matrixControlStatus object is equal to valid(1).

matrixControlTableSize

{ matrixControlEntry 3 }

The number of matrixSDEntries in the matrixSDTable for this interface. This must also be the value of the number of entries in the matrixDSTable for this interface.

matrixControlLastDeleteTime

{ matrixControlEntry 4 }

The value of sysUpTime when the last entry was deleted from the portion of the matrixSDTable or matrixDSTable associated with this matrixControlEntry. If no deletions have occurred, this value shall be zero.

matrixControlOwner

{ matrixControlEntry 5 }

The entity that configured this entry and is therefore using the resources assigned to it.

matrixControlStatus

{ matrixControlEntry 6 }

The status of this matrixControl entry. If this object is not equal to valid(1), all associated entries in the matrixSDTable and the matrixDSTable shall be deleted by the agent.

matrixSDTable

{ matrix 2 }

A list of traffic matrix entries indexed by source and destination MAC address.

matrixSDEntry

{ matrixSDTable 1 }

A collection of statistics for communications between two addresses on a particular interface:

Entry	Description
matrixSDSourceAddress	OCTET STRING
matrixSDDestAddress	OCTET STRING
matrixSDIndex	INTEGER (1 . . . 65535)
matrixSDPkts	Counter
matrixSDOctets	Counter
matrixSDErrors	Counter

matrixSDSourceAddress

{ matrixSDEntry 1 }

The source physical address.

matrixSDDestAddress

{ matrixSDEntry 2 }

The destination physical address.

matrixSDIndex

{ matrixSDEntry 3 }

The set of collected matrix statistics of which this entry is a part. The set of matrix statistics identified by a particular value of this index is associated with the same matrixControlEntry as identified by the same value of matrixControlIndex.

matrixSDPkts

{ matrixSDEntry 4 }

The number of packets transmitted from the source address to the destination address (this number includes error packets).

matrixSDOctets

{ matrixSDEntry 5 }

The number of octets (excluding framing bits but including FCS octets) contained in all packets transmitted from the source address to the destination address.

matrixSDErrors

{ matrixSDEntry 6 }

The number of error packets transmitted from the source address to the destination address.

Traffic matrix tables from destination to source always goes through destination.

matrixDSTable

{ matrix 3 }

A list of traffic matrix entries indexed by destination and source MAC address.

matrixDSEntry
{ matrixDSTable 1 }

A collection of statistics for communications between two addresses on a particular interface:

Entry	Description
matrixDSSourceAddress	OCTET STRING
matrixDSDestAddress	OCTET STRING
matrixDSIndex	INTEGER (1 . . . 65535)
matrixDSPkts	Counter
matrixDSOctets	Counter
matrixDSErrors	Counter

matrixDSSourceAddress
{ matrixDSEntry 1 }

The source physical address.

matrixDSDestAddress
{ matrixDSEntry 2 }

The destination physical address.

matrixDSIndex
{ matrixDSEntry 3 }

The set of collected matrix statistics of which this entry is a part. The set of matrix statistics identified by a particular value of this index is associated with the same matrixControlEntry as identified by the same value of matrixControlIndex.

matrixDSPkts
{ matrixDSEntry 4 }

The number of packets transmitted from the source address to the destination address (this number includes error packets).

matrixDSOctets
{ matrixDSEntry 5 }

The number of octets (excluding framing bits but including FCS octets) contained in all packets transmitted from the source address to the destination address.

matrixDSErrors
{ matrixDSEntry 6 }

The number of error packets transmitted from the source address to the destination address.

12
Filter Group

Implementation of the Filter group is optional.

The Filter group allows packets to be captured with an arbitrary filter expression. A logical data and event stream or "channel" is formed by the packets that match the filter expression.

This filter mechanism allows the creation of an arbitrary logical expression with which to filter packets. Each filter associated with a channel is ORed with the others. Within a filter, any bits checked in the data and status are ANDed with respect to other bits in the same filter. The NotMask also allows for checking for inequality. Finally, the channelAcceptType object allows for inversion of the whole equation.

The channel can be turned on or off, and can also generate events when packets pass through it.

filterTable

{ filter 1 }

A list of packet filter entries.

filterEntry

{ filterTable 1 }

A set of parameters for a packet filter applied on a particular interface:

Entry	Description
filterIndex	INTEGER (1 . . . 65535)
filterChannelIndex	INTEGER (1 . . . 65535)
filterPktDataOffset	INTEGER
filterPktData	OCTET STRING
filterPktDataMask	OCTET STRING
filterPktDataNotMask	OCTET STRING
filterPktStatus	INTEGER
filterPktStatusMask	INTEGER
filterPktStatusNotMask	INTEGER
filterOwner	OwnerString
filterStatus	INTEGER

filterIndex

{ filterEntry 1 }

An index that uniquely identifies an entry in the filter table. Each entry defines one filter that is to be applied to every packet received on an interface.

filterChannelIndex

{ filterEntry 2 }

This object identifies the channel of which this filter is a part. The filters identified by a particular value of this object are associated with the same channel as identified by the same value of the channelIndex object.

filterPktDataOffset

{ filterEntry 3 }

The offset from the beginning of each packet where a match of packet data will be attempted. This offset is measured from the point in the physical layer packet after the framing bits, if any. For example, in an Ethernet frame, this point is at the beginning of the destination MAC address.

This object may not be modified if the associated filterStatus object is equal to valid(1).

The default value for filterPktDataOffset is 0 (zero).

filterPktData

{ filterEntry 4 }

The data that is to be matched with the input packet.

For each packet received, this filter and the accompanying filterPktDataMask and filterPktDataNotMask will be adjusted for the offset. The only bits relevant to this match algorithm are those that have the corresponding filterPktDataMask bit equal to one. The following three rules are then applied to every packet:

Rule	If . . .	And . . .	Then . . .
1	The packet is too short	Does not have data corresponding to part of the filterPktData	The packet will fail this data match.
2	The bit from the packet is not equal to the corresponding bit from the filterPktData	For each relevant bit from the packet with the corresponding filterPktDataNotMask bit set to zero	The packet will fail this data match.
3	For every relevant bit from the packet with the corresponding filterPktDataNotMask bit set to one	The bit from the packet is equal to the corresponding bit from the filterPktData	The packet will fail this data match.

Any packets that have not failed any of the three matches above have passed this data match. This object may not be modified if the associated filterStatus object is equal to valid(1).

filterPktDataMask

{ filterEntry 5 }

The mask that is applied to the match process. After adjusting this mask for the offset, only those bits in the received packet that correspond to bits set in this mask are relevant for further processing by the match algorithm. The offset is applied to filterPktDataMask in the same way it is applied to the filter. For the purposes of the matching algorithm, if the associated filterPktData object is longer than this mask, this mask is conceptually extended with *1* bits until it reaches the length of the filterPktData object.

This object may not be modified if the associated filterStatus object is equal to valid(1).

filterPktDataNotMask

{ filterEntry 6 }

The inversion mask that is applied to the match process. After adjusting this mask for the offset, those relevant bits in the received packet that correspond to bits cleared in this mask must all be equal to their corresponding bits in the filterPktData object for the packet to be accepted. In addition, at least one of those relevant bits in the received packet that correspond to bits set in this mask must be different to its corresponding bit in the filterPktData object.

For the purposes of the matching algorithm, if the associated filterPktData object is longer than this mask, this mask is conceptually extended with *0* bits until it reaches the length of the filterPktData object.

This object may not be modified if the associated filterStatus object is equal to valid(1).

filterPktStatus

{ filterEntry 7 }

The status that is to be matched with the input packet. The only bits relevant to this match algorithm are those that have the corresponding filterPktStatusMask bit equal to one.

The following two rules are then applied to every packet:

1. For each relevant bit from the packet status with the corresponding filterPktStatusNotMask bit set to zero, if the bit from the packet status is not equal to the corresponding bit from the filterPktStatus, then the packet will fail this status match.

2. If for every relevant bit from the packet status with the corresponding filterPktStatusNotMask bit set to one, the bit from the packet status is equal to the corresponding bit from the filterPktStatus, then the packet will fail this status match.

Any packets that have not failed one of the two matches above have passed this status match.

The value of the packet status is a sum. This sum initially takes the value zero. Then, for each error (E) that has been discovered in this packet, 2 raised to a value representing E is added to the sum. The errors and the bits that represent them are dependent on the media type of the interface that this channel is receiving packets from.

The errors defined for a packet captured off of an Ethernet interface are as follows:

    0 — Packet is longer than 1518 octets
    1 — Packet is shorter than 64 octets
    2 — Packet experienced a CRC or Alignment error

For example, an Ethernet fragment would have a value of 6 ($2\char`^1 + 2\char`^2$).

As this MIB is expanded to new media types, this object will have other media-specific errors defined.

For the purposes of this status matching algorithm, if the packet status is longer than this object, filterPktStatus is conceptually extended with *0* bits until it reaches the size of the packet status.

This object may not be modified if the associated filterStatus object is equal to valid(1).

filterPktStatusMask

{ filterEntry 8 }

The mask that is applied to the status match process. Only those bits in the received packet that correspond to bits set in this mask are relevant for further processing by the status match algorithm. For the purposes of the matching algorithm, if the associated filterPktStatus object is longer than this mask, this mask is conceptually extended with *1* bits until it reaches the size of the filterPktStatus. In addition, if a packet status is longer than this mask, this mask is conceptually extended with *0* bits until it reaches the size of the packet status.

This object may not be modified if the associated filterStatus object is equal to valid(1).

filterPktStatusNotMask

{ filterEntry 9 }

The inversion mask that is applied to the status match process. Those relevant bits in the received packet status that correspond to bits cleared in this mask must all be equal to their corresponding bits in the filterPktStatus object for the packet to be accepted. In addition, at least one of those relevant bits in the received packet status that correspond to bits set in this mask must be different to its corresponding bit in the filterPktStatus object for the packet to be accepted.

For the purposes of the matching algorithm, if the associated filterPktStatus object or a packet status is longer than this mask, this mask is conceptually extended with *0* bits until it reaches the longer of the lengths of the filterPktStatus object or the packet status.

This object may not be modified if the associated filterStatus object is equal to valid(1).

filterOwner

{ filterEntry 10 }

The entity that configured this entry and is using the resources assigned to it.

filterStatus

{ filterEntry 11 }

The status of this filter entry.

channelTable

{ filter 2 }

A list of packet channel entries.

channelEntry

{ channelTable 1 }

A set of parameters for a packet channel applied on a particular interface:

Entry	Description
channelIndex	INTEGER (1 . . . 65535)
channelIfIndex	INTEGER (1 . . . 65535)
channelAcceptType	INTEGER
channelDataControl	INTEGER
channelTurnOnEventIndex	INTEGER (0 . . . 65535)

Entry	Description
channelTurnOffEventIndex	INTEGER (0 . . . 65535)
channelEventIndex	INTEGER (0 . . . 65535)
channelEventStatus	INTEGER
channelMatches	Counter
channelDescription	DisplayString (SIZE (0 . . . 127))
channelOwner	OwnerString
channelStatus	INTEGER

channelIndex

{ channelEntry 1 }

An index that uniquely identifies an entry in the channel table. Each entry defines one channel as a logical data and event stream.

channelIfIndex

{ channelEntry 2 }

The value of this object uniquely identifies the interface on this remote network monitoring device to which the associated filters are applied to allow data into this channel. The interface identified by a particular value of this object is the same interface as identified by the same value of the ifIndex object. The filters in this group are applied to all packets on the local network segment attached to the identified interface.

This object may not be modified if the associated channelStatus object is equal to valid(1).

channelAcceptType

{ channelEntry 3 }

This object controls the action of the filters associated with this channel. If this object is equal to acceptMatched(1), packets will be accepted to this channel if they are accepted by both the packet data and packet status matches of an associated filter. If this object is equal to acceptFailed(2), packets will be accepted to this channel only if they fail either the packet data match or the packet status match of each of the associated filters.

This object may not be modified if the associated channelStatus object is equal to valid(1).

channelDataControl

{ channelEntry 4 }

This object controls the flow of data through this channel. If this object is on(1), data, status, and events flow through this channel. If this object is off(2), data, status, and events will not flow through this channel.

The default value for channelDataControl is off.

channelTurnOnEventIndex

{ channelEntry 5 }

The value of this object identifies the event that is configured to turn the associated channelDataControl from OFF to ON when the event is generated. The event identified by a particular value of this object is the same event as identified by the same value of the eventIndex object. If there is no corresponding entry in the eventTable, then no association exists. In fact, if no event is intended for this channel, channelTurnOnEventIndex must be set to zero, a nonexistent event index.

This object may not be modified if the associated channelStatus object is equal to valid(1).

channelTurnOffEventIndex

{ channelEntry 6 }

The value of this object identifies the event that is configured to turn the associated channelDataControl from ON to OFF when the event is generated. The event identified by a particular value of this object is the same event as identified by the same value of the eventIndex object. If there is no corresponding entry in the eventTable, then no association exists. In fact, if no event is intended for this channel, channelTurnOffEventIndex must be set to zero, a nonexistent event index.

This object may not be modified if the associated channelStatus object is equal to valid(1).

channelEventIndex

{ channelEntry 7 }

The value of this object identifies the event that is configured to be generated when the associated channelDataControl is ON and a packet is matched. The event identified by a particular value of this object is the same event as identified by the same value of the eventIndex object. If there is no corresponding entry in the eventTable, then no association exists. In fact, if no event is intended for this channel, channelEventIndex must be set to zero, a nonexistent event index.

This object may not be modified if the associated channelStatus object is equal to valid(1).

channelEventStatus

{ channelEntry 8 }

The event status of this channel.

If this channel is configured to generate events when packets are matched, a means of controlling the flow of those events is often needed. When this object is equal to eventReady(1), a single event may be generated, after which this object will be set by the probe to eventFired(2). While in the eventFired(2) state, no events will be generated until the object is modified to eventReady(1) (or eventAlwaysReady(3)). The management station can thus easily respond to a notification of an event by reenabling this object.

If the management station wants to disable this flow control and allow events to be generated at will, this object may be set to eventAlwaysReady(3).

Note

Disabling the flow control is not recommended as it can result in high network traffic or other performance problems.

The default value for channelEventStatus is eventReady.

channelMatches

{ channelEntry 9 }

The number of times this channel has matched a packet. Note that this object is updated even when channelDataControl is set to OFF.

channelDescription

{ channelEntry 10 }

A comment describing this channel.

channelOwner

{ channelEntry 11 }

The entity that configured this entry and is using the resources assigned to it.

channelStatus

{ channelEntry 12 }

The status of this channel entry.

13
Packet Capture Group

Implementation of the Packet Capture group is optional.

The Packet Capture group requires implementation of the Filter group.

The Packet Capture group allows packets to be captured each time a filter is matched. The bufferControlTable controls the captured packets output from a channel that is associated with it. The captured packets are placed in entries in the captureBufferTable. These entries are associated with the bufferControlEntry on whose behalf they were stored.

bufferControlTable

{ capture 1 }

A list of buffer control entries.

bufferControlEntry

{ bufferControlTable 1 }

A set of parameters that control the collection of a stream of packets that have matched filters:

Entry	Description
bufferControlIndex	INTEGER (1 . . . 65535)
bufferControlChannelIndex	INTEGER (1 . . . 65535)
bufferControlFullStatus	INTEGER
bufferControlFullAction	INTEGER
bufferControlCaptureSliceSize	INTEGER
bufferControlDownloadSliceSize	INTEGER
bufferControlDownloadOffset	INTEGER
bufferControlMaxOctetsRequested	INTEGER
bufferControlMaxOctetsGranted	INTEGER
bufferControlCapturedPackets	INTEGER
bufferControlTurnOnTime	TimeTicks
bufferControlOwner	OwnerString
bufferControlStatus	INTEGER

bufferControlIndex

{ bufferControlEntry 1 }

An index that uniquely identifies an entry in the bufferControl table. The value of this index shall never be zero. Each entry defines one set of packets that is captured and controlled by one or more filters.

bufferControlChannelIndex

{ bufferControlEntry 2 }

An index that identifies the channel that is the source of packets for this bufferControl table. The channel identified by a particular value of this index is the same as identified by the same value of the channelIndex object.

This object may not be modified if the associated bufferControlStatus object is equal to valid(1).

bufferControlFullStatus

{ bufferControlEntry 3 }

This object shows whether the buffer has room to accept new packets or if it is full.

If . . .	And . . .	Then . . .
The status is spaceAvailable(1)	–	The buffer is accepting new packets normally.
The status is full(2)	The associated bufferControlFullAction object is wrapWhenFull	The buffer is accepting new packets by deleting enough of the oldest packets to make room for new ones as they arrive.
The status is full(2)	The bufferControlFullAction object is lockWhenFull	The buffer has stopped collecting packets.

When this object is set to full(2), the probe must not later set it to spaceAvailable(1) except in the case of a significant gain in resources such as an increase of bufferControlOctetsGranted. In particular, the wrap-mode action of deleting old packets to make room for newly arrived packets must not affect the value of this object.

bufferControlFullAction

{ bufferControlEntry 4 }

Controls the action of the buffer when it reaches the full status. When in the lockWhenFull(1) state a packet is added to the buffer that fills the buffer, the bufferControlFullStatus will be set to full(2) and this buffer will stop capturing packets.

bufferControlCaptureSliceSize
{ bufferControlEntry 5 }

The maximum number of octets of each packet that will be saved in this capture buffer. For example:

If . . .	And . . .	Then . . .
A 1500 octet packet is received by the probe	This object is set to 500	Only 500 octets of the packet will be stored in the associated capture buffer.
This variable is set to 0	–	The capture buffer will save as many octets as is possible.

This object may not be modified if the associated bufferControlStatus object is equal to valid(1).

The default value for bufferControlCaptureSize is 100.

bufferControlDownloadSliceSize
{ bufferControlEntry 6 }

The maximum number of octets of each packet in this capture buffer that will be returned in an SNMP retrieval of that packet. For example:

If . . .	And . . .	Then . . .
500 octets of a packet have been stored in the associated capture buffer, the associated bufferControlDownloadOffset is 0	This object is set to 100	The captureBufferPacket object that contains the packet will contain only the first 100 octets of the packet.

A prudent manager will take into account possible interoperability or fragmentation problems that may occur if the download slice size is set too large. In particular, conformant SNMP implementations are not required to accept messages whose length exceeds 484 octets, although they are encouraged to support larger datagrams whenever feasible.

The default value for bufferControlDownloadSliceSize is 100.

bufferControlDownloadOffset

{ bufferControlEntry 7 }

The offset of the first octet of each packet in this capture buffer that will be returned in an SNMP retrieval of that packet. For example:

If . . .	And . . .	Then . . .
500 octets of a packet have been stored in the associated capture buffer	This object is set to 100	The captureBufferPacket object that contains the packet will contain bytes starting 100 octets into the packet.

The default value for bufferControlDownloadOfset is 0 (zero).

bufferControlMaxOctetsRequested

{ bufferControlEntry 8 }

The requested maximum number of octets to be saved in this captureBuffer, including any implementation-specific overhead. If this variable is set to –1, the capture buffer will save as many octets as is possible.

When this object is created or modified, the probe should set bufferControlMax-OctetsGranted as closely to this object as is possible for the particular probe implementation and available resources. However, if the object has the special value of –1, the probe must set bufferControlMaxOctetsGranted to –1.

The default value for bufferControlMaxOctetsRequested is –1.

bufferControlMaxOctetsGranted

{ bufferControlEntry 9 }

The maximum number of octets that can be saved in this captureBuffer, including overhead. If this variable is –1, the capture buffer will save as many octets as possible.

When the bufferControlMaxOctetsRequested object is created or modified, the probe should set this object as closely to the requested value as is possible for the particular probe implementation and available resources. However, if the request object has the special value of –1, the probe must set this object to –1. The probe must not lower this value except as a result of a modification to the associated bufferControlMaxOctetsRequested object.

When this maximum number of octets is reached and a new packet is to be added to this capture buffer and the corresponding bufferControlFullAction is set to wrapWhenFull(2), enough of the oldest packets associated with this capture buffer shall be deleted by the agent so that the new packet can be added. If the corresponding bufferControlFullAction is set to lockWhenFull(1), the new packet

shall be discarded. In either case, the probe must set bufferControlFullStatus to full(2).

When the value of this object changes to a value less than the current value, entries are deleted from the captureBufferTable associated with this bufferControlEntry. Enough of the oldest of these captureBufferEntries shall be deleted by the agent so that the number of octets used remains less than or equal to the new value of this object.

When the value of this object changes to a value greater than the current value, the number of associated captureBufferEntries may be allowed to grow.

bufferControlCapturedPackets

{ bufferControlEntry 10 }

The number of packets currently in this captureBuffer.

bufferControlTurnOnTime

{ bufferControlEntry 11 }

The value of sysUpTime when this capture buffer was first turned on.

bufferControlOwner

{ bufferControlEntry 12 }

The entity that configured this entry and is using the resources assigned to it.

bufferControlStatus

{ bufferControlEntry 13 }

The status of this buffer control entry.

captureBufferTable

{ capture 2 }

A list of packets captured off of a channel.

captureBufferEntry

{ captureBufferTable 1 }

A packet captured off of an attached network:

Entry	Description
captureBufferControlIndex	INTEGER (1 . . . 65535)
captureBufferIndex	INTEGER
captureBufferPacketID	INTEGER

Entry	Description
captureBufferPacketData	OCTET STRING
captureBufferPacketLength	INTEGER
captureBufferPacketTime	INTEGER
captureBufferPacketStatus	INTEGER

captureBufferControlIndex

{ captureBufferEntry 1 }

The index of the bufferControlEntry with which this packet is associated.

captureBufferIndex

{ captureBufferEntry 2 }

An index that uniquely identifies an entry in the captureBuffer table associated with a particular bufferControlEntry. This index will start at 1 and increase by one for each new packet added with the same captureBufferControlIndex.

captureBufferPacketID

{ captureBufferEntry 3 }

An index that describes the order of packets that are received on a particular interface. The packetID of a packet captured on an interface is defined to be greater than the packetIDs of all packets captured previously on the same interface. As the captureBufferPacketID object has a maximum positive value of $2^{31} - 1$, any captureBufferPacketID object shall have the value of the associated packet's packetID mod 2^{31}.

captureBufferPacketData

{ captureBufferEntry 4 }

The data inside the packet. It starts at the beginning of the packet plus any offset specified in the associated bufferControlDownloadOffset, including any link level headers.

The length of the data in this object is . . .	Minus . . .	
The minimum length of the captured packet	The offset	
The length of the associated bufferControlCaptureSliceSize	The offset	And the associated bufferControlDownloadSliceSize.

If this minimum is less than zero, this object shall have a length of zero.

captureBufferPacketLength

{ captureBufferEntry 5 }

The actual length (off the wire) of the packet stored in this entry, including FCS octets.

captureBufferPacketTime

{ captureBufferEntry 6 }

The number of milliseconds that had passed since this capture buffer was first turned on when this packet was captured.

captureBufferPacketStatus

{ captureBufferEntry 7 }

A value which indicates the error status of this packet.

The value of this object is defined in the same way as filterPacketStatus. The value is a sum. This sum initially takes the value zero. Then, for each error (E) that has been discovered in this packet, 2 raised to a value representing E is added to the sum.

The errors defined for a packet captured off of an Ethernet interface are as follows:

0   Packet is longer than 1518 octets

1   Packet is shorter than 64 octets

2   Packet experienced a CRC or Alignment error

3   First packet in this capture buffer after it was detected that some packets were not processed correctly

For example, an Ethernet fragment would have a value of 6 ($2^1 + 2^2$).

As this MIB is expanded to new media types, this object will have other media-specific errors defined.

14
Event Group

Implementation of the Event group is optional.

The Event group controls the generation and notification of events from this device. Each entry in the eventTable describes the parameters of the event that can be triggered. Each event entry is fired by an associated condition located elsewhere in the MIB. An event entry may also be associated with a function elsewhere in the MIB that will be executed when the event is generated. For example, a channel may be turned on or off by the firing of an event.

Each eventEntry may optionally specify that a log entry be created on its behalf whenever the event occurs. Each entry may also specify that notification should occur by way of SNMP trap messages. In this case, the community for the trap message is given in the associated eventCommunity object. The enterprise and specific trap fields of the trap are determined by the condition that triggered the event. Three traps are defined in PROBEwatch: risingAlarm, fallingAlarm, and packetMatch. If the eventTable is triggered by a condition specified elsewhere, the enterprise and specific trap fields must be specified for traps generated for that condition.

eventTable

{ event 1 }

A list of events to be generated.

eventEntry

{ eventTable 1 }

A set of parameters that describe an event to be generated when certain conditions are met:

Entry	Description
eventIndex	INTEGER (1 . . . 65535)
eventDescription	DisplayString (SIZE (0 . . . 127))
eventType	INTEGER
eventCommunity	OCTET STRING (SIZE (0 . . . 127))
eventLastTimeSent	TimeTicks
eventOwner	OwnerString
eventStatus	INTEGER

eventIndex

{ eventEntry 1 }

An index that uniquely identifies an entry in the event table. Each entry defines one event that is to be generated when the appropriate conditions occur.

eventDescription

{ eventEntry 2 }

A comment describing this event entry.

eventType

{ eventEntry 3 }

The type of notification that the probe will make about this event. In the case of log, an entry is made in the log table for each event. In the case of snmp-trap, an SNMP trap is sent to one or more management stations.

eventCommunity

{ eventEntry 4 }

If an SNMP trap is to be sent, it will be sent to the SNMP community specified by this octet string. In the future, this table will be extended to include the party security mechanism. This object shall be set to a string of length zero if that mechanism is to be used to specify the destination of the trap.

eventLastTimeSent

{ eventEntry 5 }

The value of sysUpTime at the time this event entry last generated an event. If this entry has not generated any events, this value will be zero.

eventOwner

{ eventEntry 6 }

The entity that configured this entry and is using the resources assigned to it.

If this object contains a string starting with "monitor" and has associated entries in the log table, all connected management stations should retrieve those log entries, as they may have significance to all management stations connected to this device.

eventStatus

{ eventEntry 7 }

The status of this event entry.

If this object is not equal to valid(1), all associated log entries shall be deleted by the agent.

logTable

{ event 2 }

A list of events that have been logged.

logEntry

{ logTable 1 }

A set of data describing an event that has been logged:

Entry	Description
logEventIndex	INTEGER (1 . . . 65535)
logIndex	INTEGER
logDescription	DisplayString (SIZE (0 . . . 255))

logEventIndex

{ logEntry 1 }

The event entry that generated this log entry. The log identified by a particular value of this index is associated with the same eventEntry as identified by the same value of eventIndex.

logIndex

{ logEntry 2 }

An index that uniquely identifies an entry in the log table among those generated by the same eventEntries. These indexes are assigned beginning with 1 and increase by one with each new log entry. The association between values of logIndex and logEntries is fixed for the lifetime of each logEntry. The agent may choose to delete the oldest instances of logEntry as required because of lack of memory. It is an implementation-specific matter as to when this deletion may occur.

logTime

{ logEntry 3 }

The value of sysUpTime when this log entry was created.

logDescription

{ logEntry 4 }

An implementation dependent description of the event that activated this log entry.

A
Specifications and Parts

This appendix provides the following specifications for DECpacketprobe 90:

- Physical dimensions
- Environmental specifications
- Electrical specifications
- Console connector pin-out

The appendix also lists the replacement parts for DECpacketprobe 90.

A.1 Dimensions

The following table lists the physical dimensions of DECpacketprobe 90:

Dimension	Measurement
Height	3.5 cm (1.4 in)
Width	28.0 cm (11.0 in)
Depth	14.0 cm (5.5 in)
Weight	0.77 kg (1.7 lb)

A.2 Environmental Specifications

The DECpacketprobe 90 is designed to operate in an office environment or equipment room environment, such as telephone closets or satellite equipment rooms. It is not intended to operate in a sealed environment.

A.2.1 Operating Environment

The following table provides the operating environment specifications:

Condition	Value
Temperature	5°C to 50°C (41°F to 122°F)
Maximum rate of change	20°C/hr (36°F/hr)
Relative humidity	10% to 95% (noncondensing)
Wet-bulb temperature	32°C (90°F) maximum
Dew point	2°C (36°F) minimum
Altitude	Sea level to 2.4 km (8000 ft)
Air flow	Convectively cooled. A minimum of 10 cm (4 inches) of space must be provided on both ends of the unit for adequate air flow.

A.2.2 Shipping Environment

The following table provides the shipping environment specifications:

Condition	Value
Temperature	–40°C to 66°C (–40°F to 151°F)
Relative humidity	10% to 95% (noncondensing)
Altitude	Sea level to 4.9 km (16,000 ft)

A.3 Electrical Specifications

The standalone DECpacketprobe 90 has a separate self-contained power supply and a power cord.

A.3.1 Power Supply (H7827–BA)

The following table provides the power supply specifications:

Specification	Value
Voltage (domestic)	104 Vac to 128 Vac (nominal 120 Vac)
Voltage (international)	208 Vac to 256 Vac (nominal 240 Vac)
AC cord length	6 ft
Current at 120 volts	0.25 A
Current at 240 volts	0.125 A
Frequency	50 Hz to 60 Hz
Power consumption	9 W
Output voltage 1	5.1 Vdc
Output voltage 2	12.0 Vdc
DC cord length	8 ft
Output current	1.8 A

A.3.2 Input Voltage

The following table provides the input voltage specifications:

Specification	Value
Input voltages	4.75 to 5.25 Vdc
Input current	0.9 A

A.4 Console Connector Pin-Out (RS–232/DB9)

The following table provides the console connector pin-out:

Pin	Signal
1	DCD
2	SD
3	RD
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

A.5 Replacement Parts

The following table lists the replacement parts and order numbers for DECpacketprobe 90:

Replacement Part	Order Number
DECpacketprobe 90	DERMN–MA
Power supply, 110/220 Vac	H7827–BA
Back cover	H0342–AA

B

Documentation and Ordering Information

This appendix lists documentation that is related to DECpacketprobe 90. It also provides ordering information.

B.1 Related Documentation

You can order the following documents from Digital:

Document Title	Order Number
DEChub 90 Owner's Manual	EK–DEHUB–OM
DEChub 900 Owner's Manual	EK–DH2MS–OM
Open DECconnect Building Wiring Components and Application Catalog	EB–K2407–42
DECconnect System Planning and Configuration Guide	EK–DECSY–CG
PROBEwatch for ULTRIX User's Guide	AA–Q0GAA–TE
POLYCENTER Network Manager 200	QA–VM9AA–GZ
POLYCENTER SNMP Manager 300	QA–YUGAB–GZ

B.2 Ordering Information

You can order replacement parts and documentation by mail, phone, or electronically.

Need Help?

If you need help deciding which documentation best meets your needs, please call 1–800–DIGITAL (1–800–344–4825) and press 2 for technical assistance.

Electronic Orders

To place an order through your account at the Electronic Store, dial 1–800–234–1998, using a modem set to 2400 or 9600 baud. You must use a VT terminal or terminal emulator set at 8 bits, no parity. If you need help, call 1–800–DIGITAL (1–800–344–4825) and ask for an Electronic Store specialist.

Telephone or Direct Mail Orders

If you are from . . .	Call . . .	Or write . . .
U.S.A.	DECdirect Phone: 800–DIGITAL (800–344–4825) FAX: (603) 884–5597	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	Phone: (809) 781–0505 FAX: (809) 749–8377	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street Suite 200 Metro Office Park San Juan, Puerto Rico 00920
Canada	Phone: 800–267–6215 FAX: (613) 592–1946	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	—	Local Digital subsidiary or approved distributor

Digital Personnel

You can order documentation by electronic mail. Contact the following organizations for instructions:

If you need . . .	Call . . .	Contact . . .
Software documentation[1]	DTN: 241–3023 (508) 874–3023	Software Supply Business Digital Equipment Corporation 1 Digital Drive Westminster, MA 01473
Hardware documen-tation	DTN: 234–4325 (508) 351–4325 FAX: (508) 351–4467	Publishing & Circulation Services Digital Equipment Corporation NRO2–2/I5 444 Whitney Street Northboro, MA 01532

[1]Call to request an Internal Software Order Form (EN–01740–07).

Index