For: anilg
Printed on: Tue, Mar 7, 1995 10:12:41
From book: BTNG_TR
Document: RMON_TR
Last saved on: Tue, Mar 7, 1995 10:08:43

RMON Agent for OS/2

Neal Krawetz Ganesh Jayadevan Ramesh Narayan Charlie Richardson Paul Brutch
Kyle Walzel Vance Swaggerty Anil Gurijala Anish Karmarkar Ganesha Beedubail
Sridhar Muppidi Sunil Santha Willis Marti Udo Pooch

February 28, 1995

Abstract. *A remote monitoring (RMON) agent allows system management tools access to subnets that they are not immediately connected to. The RMON Agent must be capable of processing many packets with various protocols, both destined and not destined for the RMON Agent. This document describes the necessary components of a fully remotely–configurable RMON Agent for OS/2*[1].

Key Words. Network Monitoring, MIB, Client Server Model, Distributed Computing, RMON Agent.

1 Introduction

For efficient network management and distributed system analysis, the network supervisor must be capable of retrieving network statistics, such as number of packets of a particular length, as well as capturing packets from specific hosts for analysis. Problems occur when packets and statistics must be received from a remote subnet, not immediately physically available. Network traffic in one subnet is frequently not passed to other subnets, making direct monitoring impossible. An RMON Agent is a network database management system (DBMS) that actively collects network statistics from a local subnet. This data is collected in the *management information base* (MIB) and may be remotely requested by an authorized host. In Request For Comments(RFC) 1271, Remote Network Monitoring (RMON) MIB for Ethernet, and in RFC 1213, the Management Information Base for Network Management of TCP/IP–Based Internets: MIB II are given.

Dedicated RMON agents are available commercially. These machines may be placed on remote subnets and forward requested statistics to the network supervisor. Unfortunately, these machines are often expensive and, as dedicated machines, may not be used for anything but remote monitoring.

In contrast, general purpose computers with network interfaces may run software that transforms the machine into an RMON agent. This type of software may be installed when necessary, and on any subnet; allowing existing hardware to be used in place of a dedicated RMON agent. Under mul-

1. This project was supported by IBM, Austin under contract number 32525–42640–CS.

titasking operating systems such as OS/2 and Unix™, the same hardware may be used productively as an RMON agent and a workstation.

2 RMON Agent for OS/2

The design of a fully RFC 1271 and RFC 1213 compliant, software driven, remote monitoring (RMON) agent requires five major components, as shown in Figure 1. The design of this architecture, as well as the software structure for OS/2 is described in the following sections.

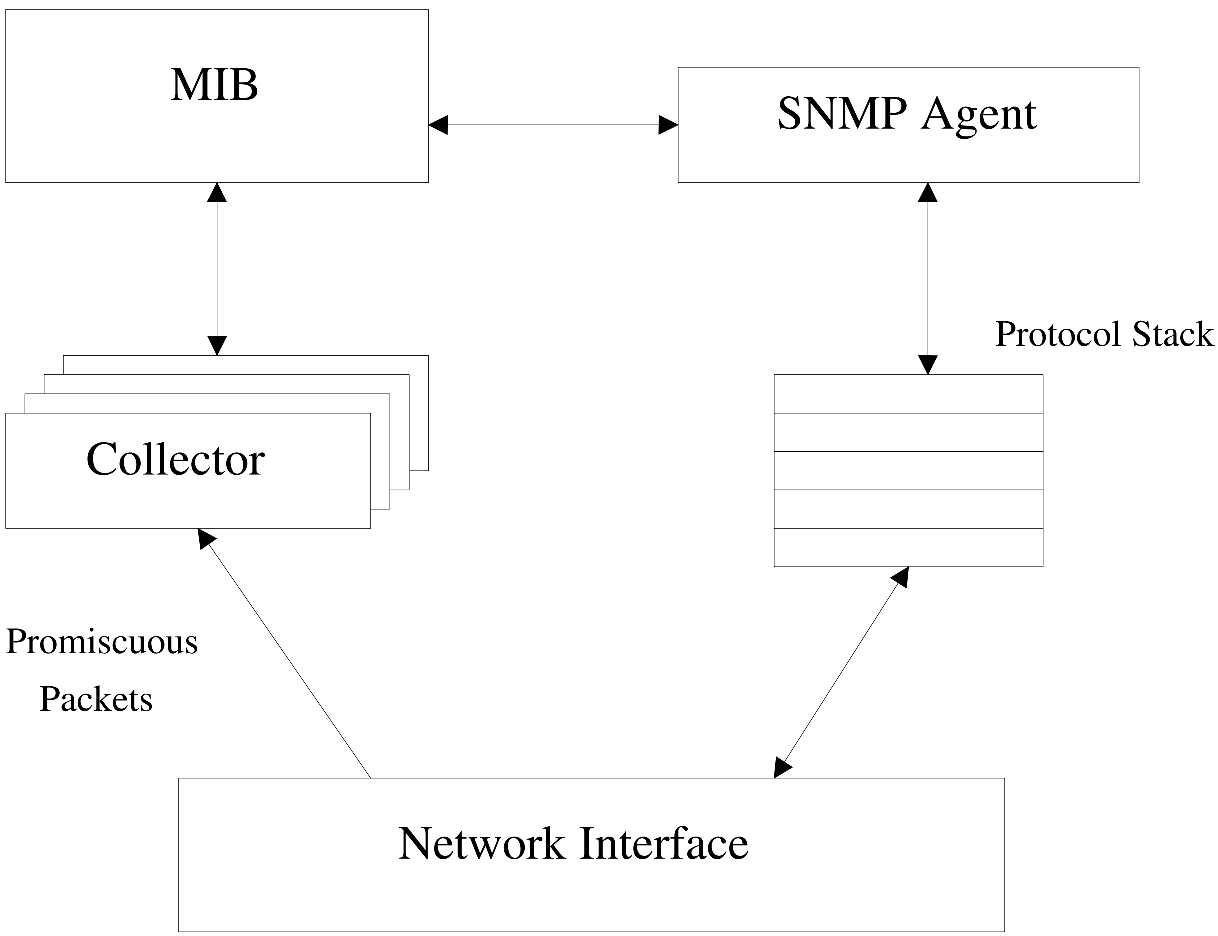

Figure 1. Major Components of RMON Agent.

3 Architecture Components

The Network Interface is used to receive and transmit packets across the available media, such as ethernet, token ring, or FDDI. For the RMON agent to work as a simple network monitoring protocol (SNMP) agent, it must respond to requests from remote servers. Thus, the network interface must allow the RMON agent to receive and transmit across the media. For collecting network statistics and packets, the interface must also allow the agent to "see" all packets on the media. This is referred to as ”promiscuous mode” since the agent may act upon any packet, regardless of the destination, and not just the packets bound for the RMON agent.

The Collector is a set of processes for maintaining network statistics and capturing packets. This component is configured by a database called the MIB (Management Information Base) and must process the promiscuous packets. The results of the processing, including counters and collected packets, are stored back in the MIB. The MIB is a DBMS, containing a large abstract data type (data) as well as functions to manage the data. These functions include querying, initializing, updating, and deleting values, as well as trigger functions so that important data added to the MIB may immediately call processing routines to handle the data. Due to the possibility of bulk processing, especially during heavy network traffic, the Collector and its MIB accesses are the priority operation of the RMON agent.

Packets that are bound for the RMON agent are not processed by the Collector. These packets are queued in the Protocol Stack for sequential processing. As packets pass through the Protocol Stack, each protocol layer is stripped and the packet is interpreted. The Protocol Stack is designed like the ISO OSI network layers; each layer contains a protocol and must be removed before interpreting the next protocol. Packets that have been processed by the RMON agent are then returned to the network via the protocol stack, where they are ”put back” together.

The Protocol Stack activates many functions for interpreting the packet. These include ARP request functions, PING requests, and other various ICMP calls. It must also respond to SNMP requests. These are handled by the SNMP Agent, which is used to set values to, and query values from, the MIB.

The MIB is the main component of the RMON agent. It is a large dynamic database of network traffic statistics, as well as a method for configuring the RMON agent. Because a remote server cannot communicate directly with the Collector (all packets are processed, but not interpreted by the Collector), the MIB must be capable of configuring the Collector. This is done by triggering specific Collector routines when certain MIB active variables are set. These active MIB variables are described in RFC 1213, and RFC 1271, and include setting packet filters, channels, and events. All MIB variables defined in RFC 1271 for Ethernet and RFC 1213 for MIB II are implemented, as well as the packet capturing method for token–ring, described in RFC 1513, in which the RIF is fully expanded. Other MIBs, including the Network Applications Technology, Inc. NAT–MIB, CISCO–MIB83 (ISO 3.6.1.4.1.9), Sun–MIB, and the Unix–MIB from RFC 1155 are completely implemented.

4 Software Structure

The software design of the RMON Agent consists of a stack of separate agent levels, as shown in Figure 2. This design allows the system to be expanded without rewriting the entire RMON Agent.

The lowest level is the actual network connection and the network interface adapter. This is hardware that, although used by the system, is implementation transparent. Thus, twisted pair, coax, or radio medium, as well as ethernet, token ring, or FDDI hardware is independent of the RMON agent.

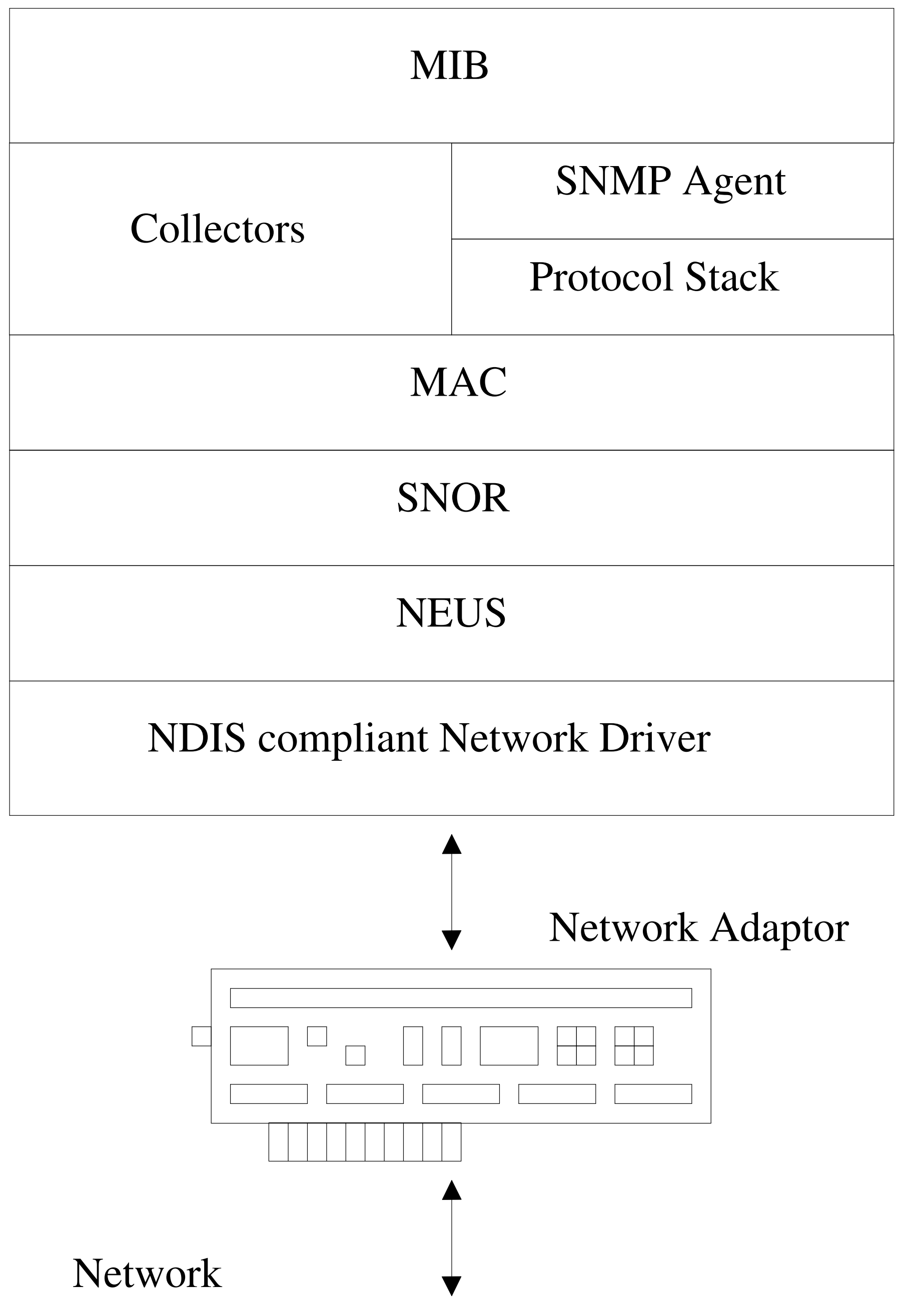

Figure 2. Layers of RMON Agent Software.

The RMON agent communicates with the network adapter by means of a network driver interface specification [NDIS] compliant network driver. This driver allows a generic method for controlling the hardware. Setting the hardware to promiscuous mode, receiving, and transmitting are handled by the driver. As long as it is a fully implemented NDIS driver, the RMON agent does not need to be modified for different types of networks.

Above the NDIS driver are the specific components built for the RMON agent. The network universal system (NEUS) device driver is used to interface with the network driver. In OS/2, this is the NDIS driver, but for other machine platforms, such as AIX or SunOS, this is a component of the operating system. The NEUS driver is operating system dependent, but allows the rest of the RMON agent to be very portable by providing a standard interface. In OS/2, the NDIS driver cannot initiate a function call for handling specific events. The NEUS device driver interfaces with the NDIS driver and keeps track of network statistics (number of collisions, CRC errors, etc.) and sets a semaphore that is interpreted by the RMON Agent each time a packet has been received for processing. Because the NEUS driver directly ties into the NDIS driver, no other network access may be performed; the computer system becomes a dedicated RMON agent for all network purposes.

The statistics network routine (SNOR) is the lowest part of the RMON Agent that is not a device driver. This component of the executable manages the network statistics provided by the NEUS driver, as well as receives the packets from, and sends packets to, the NEUS driver. Received packets are stripped of their NDIS and NEUS information, and are passed to the medium access control (MAC) layer, where the hardware specific protocols are identified. This is necessary since different media types contain different length header information. The identified packet is then passed along two independent paths.

The Collector receives the packet and performs two tasks: compute statistics and capture packets. The statistics include the various MIB variables for counting octets, the number of valid packets seen, and the number of packets that matched (or did not match) the active filters and channels. The capture component is used for storing packets for later recall from a remote host. The packet identification performed by the MAC is used by the Collector for interpreting different field lengths and handling network specific headers. For example, the token ring header must contain a fully expanded routing information field (RIF) before capturing; the token ring packet is identified as such by the MAC layer.

The other path from the MAC contains the Protocol Stack and the SNMP Agent. As the packet traverses the Protocol Stack, the layers of information are processed and the packet may be removed from the stack at any time. For example, the first protocol is the network independent MAC header; packets not destined for the RMON Agent are removed. ARP packets are then identified, responded to, and removed from the stack. TCP headers are interpreted, followed by IP headers, and so on, until the SNMP data is left. The SNMP data is passed to the SNMP Agent for processing.

The SNMP agent is used for remote modification and retrieval of the MIB entries. Each access to the MIB is limited by a community string (password) and host address. Four types of access are permitted: read–only, write–only, read–write, and none. Each community string has an associated access privilege and may contain a list of permissible host IP addresses. Invalid access attempts remove the packet from further processing. Valid requests are responded to according to the desired request. SNMP requests transmit a MIB variable to access, the type of access, and any necessary data. Requests that affect the Collector, such as the modification of a packet filter, cause the MIB to call the Collector with the appropriate change.

All outbound packets from the RMON Agent must traverse the Protocol Stack (where the packets are appended with the appropriate protocol information), MAC, SNOR, NEUS, and NDIS layers before reaching the network adapter.

5 Initialization

The RMON Agent performs a five–stage initialization. The initialization primes the system for network communication and initializes the agent with default values. The first stage in the initialization is the NEUS low–level configuration. Because NEUS is a device driver, this stage occurs only once, at system boot–up. When the OS/2 system boots, the NDIS driver is loaded and initializes the interface adapter; the NEUS driver is then loaded and binds to the NDIS driver. This binding registers the NEUS for packet and statistical communications with the network adapter. After initialization, all packets seen by the adapter and all errors found (CRC errors, collisions, etc.) are passed from the NDIS driver to the NEUS driver for handling.

The second stage in the initialization process occurs each time the RMON agent executable is started. The executable registers itself with the NEUS driver. This process registers a semaphore handle with the NEUS driver. The semaphore is triggered by the NEUS driver each time information is available for processing; use of a semaphore allows the executable to perform other tasks rather than remain in a constant polling loop. The NEUS device driver also contains a set of device driver commands for communicating with the NDIS driver. Applications are not expected to communicate directly with the NDIS driver since they may invalidate the information processed by NEUS.

After registering the semaphore, the executable initializes the network adapter through the NEUS device driver commands. The adapter is set to promiscuous mode and reinitialized if necessary (the NDIS driver contains a set of flags that state whether promiscuous mode is allowed and whether a reinitialization is necessary). If this initialization fails (i.e., no promiscuous mode is allowed), the RMON Agent displays an error message and aborts all NDIS communications.

The forth stage registers the available protocols in the Protocol Stack. Each protocol contains three functions: INITIALIZE, ENCODE, and DECODE. The INITIALIZE functions create an entry in the protocol array and informs the system that the protocol is available. Some functions may not be necessary for some configurations, such as ethernet on a token ring system; the INITIALIZE function verifies that the ENCODE and DECODE functions will be able to perform properly. The protocol array contains a list of valid ENCODE and DECODE functions with their associated protocol index. Each new packet is compared to the protocol index and processed with the proper ENCODE/DECODE functions.

The final initialization stage sets up the MIB information. Because the MIB is an active database, capable of initiating function calls based on stored values, the default MIB values are stored in an external configuration file and are user configurable. These defaults include the RMON Agent's IP address, default router, and network mask, as well as available community strings and hosts. Other configurations items include starting initial collectors or performance monitors and SNMP Trap startup notification (if available).

6 The Collector

All packets received by the RMON Agent are processed by the Collector. The processing of the packet is as follows:

1. The NDIS (in promiscuous mode) receives the packet.
2. The NEUS driver receives the packet from the NDIS driver.
3. The SNOR unit identifies bad packets, caused by collisions or bad CRC's and modifies the MIB accordingly. The number of octets in the packet are also recorded in the MIB.
4. The MAC receives the valid packet and identifies the protocol type for network statistics in the MIB. For example, the number of UDP packets seen are stored. Each available protocol contains a VALID function; if the function returns TRUE, then the packet is VALID for that protocol. Each ISO OSI protocol layer is identified in order for proper validation. Thus, the physical layer is identified first, followed by the network layer, and so on. This process does not actually modify the packet, but does identify the packet's component parts.
5. The complete packet is passed the Collector, where it is filtered and stored.

The Collector unit consists of three components: channel, filter, and collector [RFC 1213]. Each interface to the RMON Agent is called a channel; an interface adapter may have more than one channel active at a time; this allows multiple hosts to use the RMON Agent concurrently. To capture a packet, a channel must be created and activated in the MIB by a remote host. The Collector queries the MIB for the available channels and their status'. For each channel that is active, a set of filters may exist. These filters are used to narrow the selection of packets to capture.

Filters consist of a set of filter bytes, a set of mask bytes, and an offset size. Each set of filter bytes has an associated mask byte. The packet bytes are compared to the filter, starting at the offset. For each mask bit that is non–zero, the value within the packet's bit must match the value in the filter. If the packet does not match the filter, or is smaller than the filter, then the packet is rejected. Packets that are longer than the filter, but that match the filter at the offset, are accepted. Multiple filters on a channel are applied in a logical OR fashion, while the bits within a filter are applied in a logical AND fashion.

The Collector unit compares each packet to each active channel's active filter. The packets that do not match the filter(s) are rejected from the system and requires no more processing. Accepted packets are collected and stored in the MIB. The collection process does not store the entire packet. The MIB contains an offset and size for the collection; for accepted packets, the segment of packet bytes starting at the collection offset and no larger than then collection size is stored in the MIB channel buffer. The size of the channel buffer is stored another MIB variable, as well as what to do when the buffer fills (wrap when full or stop when full). Packets that are stored in the MIB require no more processing.

Some adapter hardware (i.e., token ring) are full duplex, while others (i.e., ethernet) are half–duplex. All packets received by the half–duplex adapters are processed without consideration, but the full–duplex adapters may cause feed–back in the system. Because the RMON Agent sees what it transmits in full–duplex, a race condition may occur when the RMON Agent captures what it transmits. This forces the capture buffer to never empty, and may capture more of it's own packets than promiscuous network traffic. As a solution, packets identified by the MAC unit as having the same source hardware address as the RMON agent (identified by the NDIS driver) are ignored.

The creation and activation of channels, filters, and buffers are performed by the SNMP Agent.

7 SNMP Agent Communication

This section describes the processing of a packet that is directed at the RMON Agent. These packets consist of three categories: MIB–read requests, MIB–write requests, and PING requests. These transactions are used to query the MIB, set the MIB and configure the RMON Agent, and detect the presence of the RMON Agent, respectively.

Initially, each incoming packet is processed the same way as the Collector's packets: adapter to NDIS to NEUS, SNOR, and MAC; but the MAC physical layer identification may identify the packet as bound for the RMON Agent. This occurs when the hardware destination address of the packet is the same as the Agent's hardware address (identified by the NDIS driver during initialization). These packets are send to the Protocol Stack instead of the Collector.

The Protocol Stack is a series of DECODE and ENCODE functions for stripping off (and later appending) the individual protocol fields. The sequence of DECODEs performed on a packet are stored in a stack, and later recalled as the outgoing packet is prepared. All packets that successfully survive the Protocol Stack cause some type of response packet.

The first decode function is from the physical layer. The physical layer type is identified by the NDIS driver during initialization. The physical layer DECODE returns the source and destination hardware addresses, as well as the packet type and remaining data in the packet. Other physical layer information, such as the RIF (token ring routing information field) are stored for the corresponding ENCODE function. The ENCODE function appends the physical layer data to the packet and forwards the packet to the MAC unit for transmitting. (The MAC unit, in turn, forwards the packet to the SNOR and NEUS components, and finally to the NDIS and network adapter.) Ethernet and token ring are currently implemented, but other protocols such as FDDI are readily available. Throughout the Protocol Stack, packets that do not match the known protocols are rejected from the Protocol Stack and are no further processed.

After the physical layer, the packet type is decoded. These include the ARP (RARP) and IP packets. For the ARP packets, no address lookup table is necessary since the RMON Agent does not initiate network transactions[1]. The IP DECODE stores the information necessary for the associated ENCODE and further processes the packet via the Protocol Stack.

ICMP, TCP, and UDP packets are then processed. For ICMP, all types of packets are accepted and processed. The more useful of which is ECHO_REQUEST (PING) that may be used to verify the operation and proper initialization of the RMON Agent. If the host system receives the ECHO_REPLY, then it verifies that the NDIS, NEUS, SNOR, MAC, and Protocol Stack (up to this point) are properly implemented and configured.

Both TCP and UDP packets are decoded, and the destination port is identified. Currently, only the ICMP functions ECHO, DISCARD, and TIME, and SNMP are implemented. ECHO, DISCARD, and TIME initiate separate functions that perform the corresponding response. SNMP packets processed by the SNMP Agent, where they are broken down by request type. SNMP packets that do not match the known community strings and available hosts (as recalled by the MIB) are ignored and create no response; although an error code is displayed to the local console. Attempted accesses that do not match the community's access type create an invalid request response. Valid requests are performed with the MIB (write and read), and the appropriate response as defined by RFC 1213 is returned.

[1]During initialization, a SNMP–Trap request is sent to the Trap host(s). This does require an ARP request, but is only done once and does not require a lookup table.

Packets that return via the Protocol Stack are appended with the appropriate (stacked) headers and trailers for the protocol. The final valid packet is passed through the MAC and SNOR units, and finally to the NEUS driver where it is transmitted to the adapter and over the network.

The final SNMP packet transaction looks like:

1. NDIS receives the promiscuous packet from the adapter.
2. The NEUS driver signals a semaphore that a packet has arrived.
3. The SNOR unit retrieves the packet and processes the statistics for the packet.
4. The MAC unit identifies the packet as bound for the Protocol Stack.
5. The Protocol Stack begins the decoding process.
6. The link layer is decoded and MAC packet type is identified as IP.
7. The IP packet is decoded and the following protocol is identified as UDP (or TCP).
8. The UDP (or TCP) header is decoded and the destination port is identified as SNMP.
9. The SNMP packet is passed to the SNMP Agent.
10. The SNMP Agent performs the SNMP request with the MIB (read or write).
11. The reply SNMP packet is created and passed back through the Protocol Stack.
12. The UDP (or TCP) header is appended to the packet.
13. The IP header is appended to the packet.
14. The link layer header, and possible trailer, are appended to the packet.
15. The packet is forwarded through the MAC and SNOR units without interpretation.
16. The NEUS driver receives the packet and forwards it to the NDIS driver.
17. The NDIS driver transmits the packet over the interface adapter.

8 Differences Among RMON Agents

The OS/2 RMON Agent differs from other non–OS/2 agents in a variety of ways. The available Unix RMON agents do not require an NDIS or NEUS device driver; forcing the network interface tap and RMON agent to be hardware dependent. Under Unix, the Protocol Stack becomes unnecessary since the operating system contains all of the necessary protocol verification components. DOS RMON agents require a completely dedicated machine, while the OS/2 agent only requires a dedicated network adapter. Finally, dedicated stand–alone RMON agents, such as the Ethermeter, are relatively much faster in execution since they are specialized systems and designed, both hardware and firmware, around a single specific task.

9 Future Work

The current OS/2 RMON agent implementation includes network interface support for ethernet and token ring, but does not have an implemented the token ring MIB (RFC 1513). This will be implemented fully. Furture plans also include FDDI support. The current system is compiled for OS/2 version 1.3, but operates in OS/2 2.1 properly; a future enhancement is to port the system completely to OS/2 2.x and OS/2 Warp, in order to fully take advantace of the 32 bit operating system.

A major limitation for the RMON agent is the inability for other software to use the network interface while the RMON agent is installed. This is due to the promiscuous interface connection and the constant packet monitoring required by BTNG[Otton93]. Furture versions of the software may include either allowing other network software to operate when BTNG is not in use (without needing to reboot the system), or having BTNG pass all host addressed packets that do not contain SNMP requests to other network software, allowing the simultaneous use of the network interface between network software packages.

10 Conclusions

A remote monitoring agent is the only practical solution for monitoring network traffic over multiple subnets. By implementing the RMON agent on a PC through software, no special hardware is necessary. The RMON agent, as defined in RFC 1213, RFC 1271, and RFC 1513, allows a network manager to monitor remote packets, as well as configure the agent for special analysis and capture requirements. The modular design of the OS/2 RMON agent allows for easy expendability; future network protocols and network interfaces may be readily appended to an existing system.

References

[Otton93] Otten, H. A., The Beholder Cookbook, Delf University of Technology, Netherlands. (1993).

Request For Comment 1213, *Management Information Base for Network Management of TCP/IP–Based Internets: MIB I*I.

Request For Comment 1271, *Remote Network Monitoring (RMON) MIB for Ethernet*.

Request For Comment 1513, *Token–Ring Extensions to the RMON MIB*.

[NDIS] *LAN Manager Network Driver Interface Specification (NDIS), Microsoft and 3Com*, Version 1.0.1 (1988).