THE DESIGN AND IMPLEMENTATION OF A NETWORK MANAGEMENT APPLICATION USING SNMP PROTOCOL

By

WENXIA ZHANG

Bachelor of Science
Tianjin Finance & Monetary University
Tianjin, China
1994

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2000

THE DESIGN AND IMPLEMENTATION OF A NETWORK

MANAGEMENT APPLICATION

USING SNMP PROTOCOL

Thesis Approved:

Jacques E. LaFrance

Thesis Adviser

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to express my special gratitude to my advisor Dr. Jacques E. LaFrance for his encouragement and guidance in my entire graduate study.

I would also like to thank Dr. Jing Peng and Dr. H. K. Dai for serving as my committee members and reviewing this thesis, without their suggestions, ideas and support, it would be impossible to finish the thesis on time.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Computer networks have been growing explosively over the last two decades. Today's networks are often internetworks and intranetworks that have greatly increased in size and complexity due to the fact that network hardware and software from different vendors often coexist in the same network. It is also possible for different network architectures and technologies to coexist in these internetworks. As more and more people have access to networks, the security issue also becomes a major concern. In order to maintain network performance and efficiently use the network resources, it is very important to have a standard way of managing networks.

Over the last few years, two standards emerged from the network management community, one is the SNMP (Simple Network Management Protocol), another is ISO CMIP (Common Management Information Protocol). However, with the rapidly growth of Internet and the wide adoption of TCP/IP protocol, SNMP which is based on the TCP/IP protocol has gathered extremely widespread support among internetworking vendors of hardware, software, and network management platform products.

SNMP is an open standard so that it can operate with many different types of network devices from various vendors and can be integrated into networks of different sizes, ranging from small LANs (Local Area Networks) to large WANs (Wide Area Networks), and different existing networking technologies such as Frame Relay, ATM

(Asynchronous Transfer Mode), and so on. It is also easily expanded as newer networking technologies and equipment emerge. Furthermore, it is cost-effective: 1) its documentation is free, the standards are published as RFCs (Request for Comments), 2) it is relatively easy to implement because of its vendor-independence, 3) it can be used to manage a wide range of devices, services and applications, and 4) it can be implemented to fulfill the major functions of network management including fault management, accounting management, configuration management, performance management, security management, and more.

TCP/IP-based internetworks are designed to be multivendor systems, although a TCP/IP-based version of CMIP (known as CMOT – CMIP Over TCP/IP) is also available as RFC1189, but it has not gathered the widespread support as of SNMP. Based on the current industry consensus support of SNMP, it would be significant for the internet technology development to do some research on SNMP. It is for this reason that I choose this challenging field for my thesis study.

CHAPTER II

LITERATURE REVIEW

SNMP is the standard operations and maintenance protocol for the network and internets. SNMP is also the key technology that enabled the Internet's phenomenal growth, it is widely used in today's internet management.

Current research mainly covers two aspects of the SNMP field: SNMP protocol studies (including SNMPv1, SNMPv2 and SNMPv3) and MIB (Management Information Base) studies.

2.1 SNMP Protocols

The proposed research work on SNMP protocols presents some background information, the overview of network management requirements and an explanation of fundamentals such as network management architecture; performance, fault and accounting monitoring; and configuration and security control. To understand the history of SNMP, one need to know and check the RFCs. RFC stands for Request For Comments, it is a prime vehicle for disseminating information about the internet and its standards. The SNMP specifications in their various stages have been published as RFCs. The RFCs can be obtained through a variety of means from different sources. They are stored on-line and may be obtained at no charge over the internet using e-mail,

anonymous FTP, or a web browser. The original SNMP authors are Jeffrey Case, M. Fedor, M.L. Schoffstall, and J. Davin, their research was published in 1988 as RFC 1067 entitled "A Simple Network Management Protocol"[1]. Before the publication of RFC 1067, many others also made significant contributions to the SNMP protocol. For example, Keith McCloghrie and Marshall Rose finished the three key elements of SNMP: the SMI, the MIB, and the protocol. Their work was published in RFC 1065 "Structure and Identification of Management Information for TCP/IP-based internets"[2] and in RFC 1066 "Management Information Base for Network Management of TCP/IP-based internets"[3]. In April 1989, Schoffstall, Davin, Fedor, and Case revised the SNMP protocol and issued "Simple Network Management Protocol(SNMP)" as RFC 1098[4]. From then on, many vendors released SNMP implementations, such as Cisco, Proteon and the Wollongong Group. In 1990, the SNMP operability study was done by many scholars. Rose, McCloghrie, and Davin issued RFC 1187 on how tables can be retrieved more efficiently in SNMP, "Bulk Table Retrieval with SNMP"[5]. By 1992 the talk of SNMP Version 2 to improve the management framework was well underway. One very important magazine –Simple Times— is an excellent source of information on SNMP. It includes technical articles, the status of working groups and the SNMP RFCs, and some book reviews. Sean Harnedy gave some very detailed information about the Simple Times and the evolution of the standard SNMP[6]. In April 1993, more than 10 RFCs regarding the SNMPv2 were issued. Case, McCloghrie, Rose, and Waldbuser discussed the network management framework, the structure of SNMPv2, some conventions, the administrative model of SNMPv2, the security protocols and the transport mappings for SNMPv2. However, the implementation of the draft SNMPv2 specifications have been

slow to appear from both manager and agent vendors. Until now, although the SNMPv2 implementations are widely available from agent and manager vendors, SNMPv1 is still the most pervasive form of network management. In 1997, the SNMPv3 working group was formed to begin work on the latest version of SNMP. The research on SNMPv2 and SNMPv3 mainly focuses on security features such as message authentication code and encryption, USM (User-Based Security Model), and VACM (View-Based Access Control Model). In 1998, William Stallings provided some very good discussions on the SNMPv3 security features: authentication, privacy and access control [7, 8]. It is important to realize that SNMPv3 is not a stand-alone replacement for SNMPv1 or SNMPv2. SNMPv3 defines a security capability to be used in conjunction with SNMPv2 or SNMPv1. SNMP provided a standard internetworking management protocol. Some scholars made some research on the SNMP based management architecture for Internet Information Services. For example, F. Stamatelopoulos and B. Maglaris presented a hierarchical management scheme for Internet Information Services[9], the hierarchy consists of at least three layers: the agent, the domain manager/remote monitoring agent(DM/RMA) and the management station(MS). The management of mobile networks using SNMP protocol also caught some scholars' attention. Luca Deri and others developed a mobile network management system by the implementation of SNMP using Java[10]. Other scholars researched the SNMP Basic Encoding Rules for information packets encoding and decoding across the heterogeneous network layers [11].

2.2 Management Information Base (MIBs)

Other proposed research work has concentrated on the study of MIBs (Management Information Base). MIB specifies what variables the network elements maintain (the information that can be queried and set by the manager). Some proposed research offers a functional view of SNMP-based management, emphasizing the aspects that relate directly to MIBs. These include 1) modeling and development, 2) relationships between objects, 3) textual conventions, 4) domains and control fields, 5) versions and migration[12]. Some research has covered the basic syntax and specifications of MIB modules and the discussions of advanced data structures and data types, including 1) nested tables and multi-table relationships, 2) linked lists, 3) multidimensional arrays, 4) floating point numbers[13].

Two versions of the Internet MIB for SNMPv1 have been published : MIB-I, RFC 1156[14], and the enhanced MIB-II, RFC 1213[15]. Three excellent sources of information on MIBs include Dave Perkins' "How to Read and Use an SNMP MIB"[16], Bob Stewart's " Development and Integration of a Management Information Base"[17], and 3Com Corporation's "Introduction to Simple Network Management Protocol: A Self-Study Guide"[18]. One important MIB is RMON MIB: Remote Network Monitoring MIB. Several RFCs define RMON functions: RMON for Ethernet networks, RFC 1757[19]; RMON extensions to support upper-layer protocol functions, known as RFC 1513[20]. Some scholars such as Kevin Tolly and Jim Carr discussed some RMON applications[21].

RFC 1155 specified the Structure of Management Information (known as SMI) which is a set of common structures and an identification scheme used to reference the

variables in the MIB[22]. For example, the SMI specifies that a Counter is a nonnegative integer that counts from 0 through 4,294,967,295.

There are several commercial network management applications available that are based on SNMP, such as the OpenView from Hewlett-Packard and the NetView from IBM/Tivoli. Utilities such as the WinSNMP APIs and SNMP++ classes set also exist that help network engineers develop network management applications.

However, all the commercial applications have some disadvantages. First of all, they are very expensive, an enterprise version of HP OpenView will cost almost $20,000 to install and for each additional user there will be more license fees to be charged, plus, you need pay the expensive service and consulting fees. Besides the cost, these applications can not resolve all the requirements of the network management, for example, this commercial software cannot resolve some complex network security issues and often the business needs some duplicate investment in the security management of the network. For a middle or small sized network, it is unaffordable to purchase this kind of software to manage the network. However, because SNMP is an open standard, it is possible to develop some specific applications for the specific needs of any network management. As Microsoft Visual Basic (VB) and Visual C++ (VC++), especially the ActiveX controls, become more and more popular in developing GUI applications, it would be very beneficial to have a custom control that could help in developing network management applications in VB or VC++. This is exactly the goal of this thesis: create an SNMP ActiveX control, called ***SnmpPacket*** control, and then use it to develop a simple network management application called ***vbsnmp*** – a network manager, we can use this network manager (***vbsnmp***) to demonstrate how flexible and expandable it is to fetch and

set values of managed objects from a given SNMP agent. By doing so, it shows how this kind of research can benefit the management of middle and small sized networks.

CHAPTER III

THE ARCHITECTURE OF SNMP

3.1 The Manager/Agent Model

Over the past decade, computing architectures have moved from centralized, mainframe-based environments to distributed Client/Server environments. Similarly, the network management systems architecture has moved from host-based systems to distributed systems which use a methodology called the Manager/Agent Model. The network devices managed by the Manager/Agent Model systems are called objects. The definition for each object is contained in the Management Information Base, which will be discussed shortly. The Agent resides in the object and reports the object's current status to the Manager. Most internetworking products, from the simplest bridge to the most complex ATM switch, come with an embedded SNMP agent. The Manager maintains global knowledge of the internetwork in question. The Manager is the application system to be developed to manage the network device which supports the SNMP Agent. The Manager usually includes three functions: a Graphical User Interface (GUI), a database, and communication facilities. The GUI allows end users to visualize the internetwork, the database keeps track of the internetwork elements and parameters for those elements, and the communication facilities which use SNMP protocol help the Manager to communicate with the managed elements. These three functions enable the Manager to see what is happening on the network, to communicate with the devices being

managed to query or modify parameters, and to oversee dynamically the network operations.

SNMP uses the Manager/Agent Management Model to monitor and control a network. Figure 3.1 shows the relationship between the Manager/Agent Management Model and the TCP/IP 4-layer communication model.

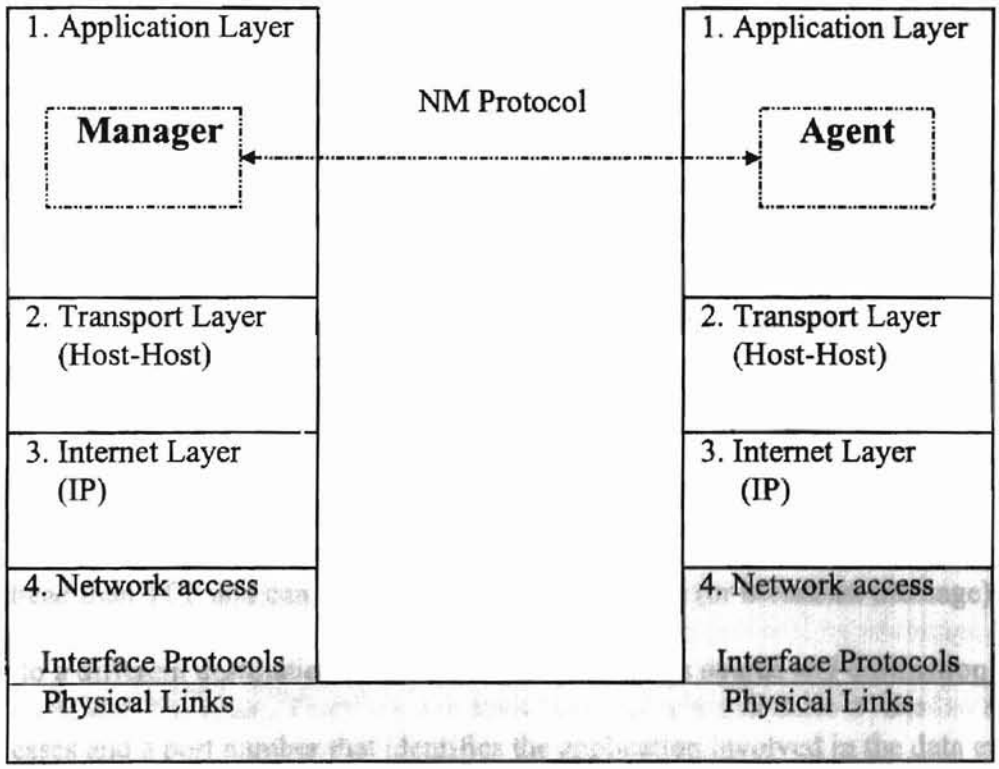

Figure 3.1 Manager/Agent Management Model

The figure depicts two network devices, one is termed the network management *manager*, and the other is termed the network management *agent*. Both the manager and the agent operate at the application level (the top layer) in the TCP/IP 4-layer communication model. They communicate with each other to exchange management information via a network management protocol which, in this case, is the SNMP.

The SNMP protocol is defined as an application-level protocol. It relies on a set of lower-level protocols and device drivers for the networking communications. When available, the TCP/IP protocol suite is adapted to realize the communications between the manager and agent. Figure 3.2 shows how SNMP and the primary protocols in the TCP/IP protocol suite fit into a typical network management message. Notice that SNMP uses the User Datagram Protocol (UDP) as the transport service.

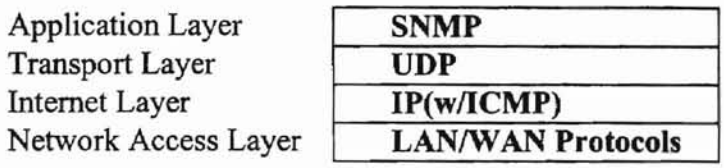

Figure 3.2 SNMP And The TCP/IP Protocol Suite

The UDP is a connectionless transport protocol that can send data without a pre-established data-circuit connection. The advantage of UDP over TCP is that UDP has less overhead than TCP and can send a sequence of messages (or broadcast message) with each to a different destination. Each UDP data packet has source and destination IP addresses and a port number that identifies the application involved in the data exchange. For SNMP, this port number is usually 161, if the agent replies back any network error, it will send traps back to manager side UDP port 162. By using two different port numbers, a single system can easily run both a manager and an agent. The disadvantage of the UDP is that it provides no reliability: it sends the datagrams that the application writes to the IP layer, but there is no guarantee that they ever reach their destination. This means that a request from the manager may not arrive at the agent, and the agent's reply may not make

it back to the manager. The manager probably wants to implement a timeout and retransmission. However, since SNMP messages only have five commands which we will discuss soon and four of them are simple request-reply protocols (the manager sends a request, the agent sends back a reply), using UDP will be good enough for the SNMP.

The Manager is the active participant in the SNMP model. In the general case, the Manager initiates requests to manage one or more network nodes. The agents passively wait for requests from the Manager. When an agent receives a request, it translates the request, performs necessary operation(s), sends an appropriate response, and then goes back to the "waiting" state.

3.2 The Fetch-Store Paradigm

The SNMP model uses the *fetch-store* paradigm for interaction between a manager and an agent. Each agent, also called an SNMP server, maintains a collection of data associated with all of its network devices and services, called *managed objects*. The Manager, also called the SNMP client, can read or write these data by exchanging SNMP messages with the agent. There are two basic types of operations: *fetch*, used to read a value from an object, and *store*, used to write a value to an object. By fetching values of managed objects, the manager can determine their status. Storing a value to an object causes a change of the object's setting or causes the agent to perform a certain predefined operation such as a device reboot.

The SNMP version 1 (SNMPv1) has only five commands for management:

- `get-request`
- `set-request`

- `get-next-request`
- `get-response`
- `trap`

The Manager can fetch values of objects by sending `get-request` or `get-next-request` messages. The former fetches the value of an object specified in the request message while the latter reads the value of the next valid object managed by the agent. This provides a convenient way of tree-traversing all the objects the agent manages. The `set-request` command is used by the Manager to modify the value of a specific object. The agent can reply to these three commands with the `get-response` message if no error occurs. The agent is also capable of sending a message to Manager notifying the occurrence of some predefined condition. The Manager is then responsible for determining any future interaction between the agent and subsequent action(s).

In SNMP version 2 (SNMPv2), two additional messages have been added: one is called `get-bulk-request` message, which allows the Manager to fetch large amounts of data with a single request, the other is called `inform-request` message and is used for manager-manager communications.

3.3 Management information base (MIB)

The collection of all the object types SNMP can manage is known as the Management Information Base or MIB. Each object type in the MIB is defined by a set of rules called the Structure of Management Information (SMI) using the ISO Abstract Syntax Notation One (ASN.1) protocol, and possesses three basic attributes: *object type name*, *object type syntax*, and *object type encoding*.

The object type name, also called *object identifier*, is a unique representation for identifying an object type in the MIB. In order for the naming system to be flexible and expandable, SNMP adapts the ASN.1 naming convention so that every object in the MIB is placed in a global tree that begins with an unnamed root and has nodes appended to it to represent the various objects. Each node is assigned a number and the subsequent sibling nodes are assigned incremental numbers. Each object identifier is represented as a string of these numbers separated by periods. Figure 3.3 shows part of the MIB tree that is of interest for SNMP.

As an example of the object identifier, the first object under the system group whose value is a string that describes the managed network has a name:

```
iso.org.dod.internet.management.mib2.system.sysDescr
```

or

```
1.3.6.1.2.1.1.1
```

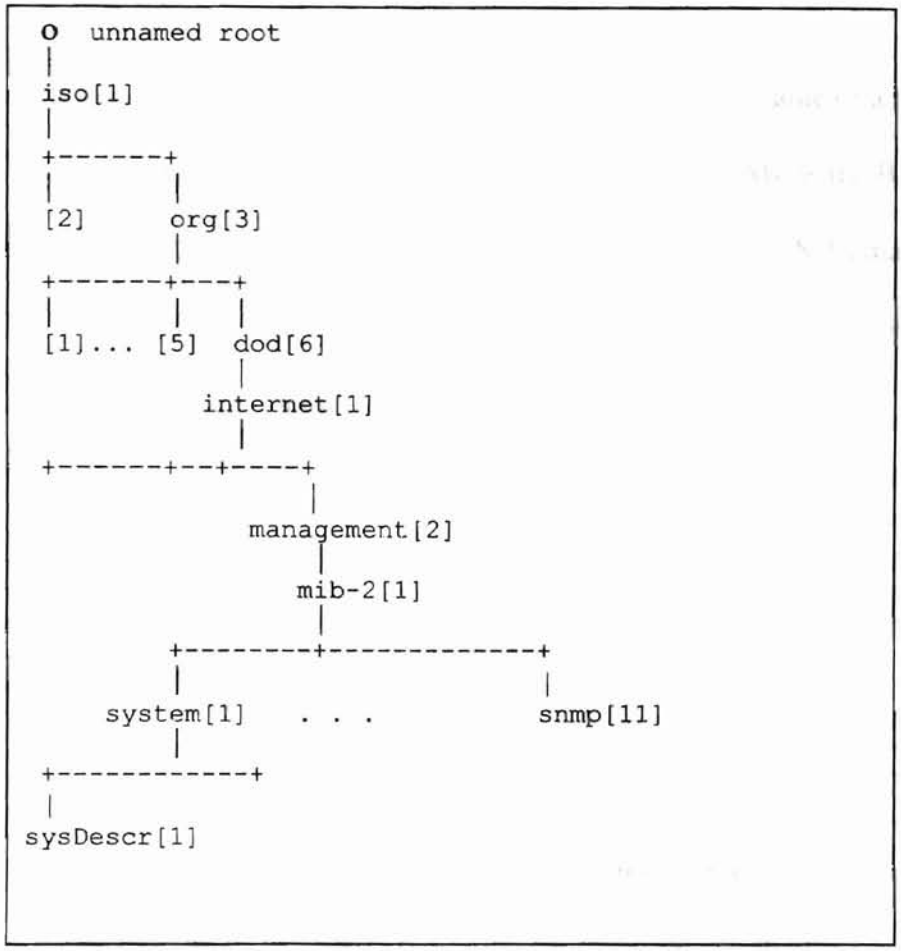

Figure 3.3 Object Identifiers In The Management Information Base

The object type syntax defines the abstract data structure that describes the object. The data structure specifies four properties of the object:

- **Type**, the type of the object such as integer, octet string, IP address or other type.
- **Access mode**, the permission level that the agent examines when a request is received for the object.
- **Status**, the managed node's responsibility for implementing the object. Mandatory or optional?
- **Name value**, a textual name that is equivalent to its object identifier.

The SMI also specifies the encoding rules for transmitting the value of a managed object between a manager and an agent. The encoding rule for SNMP is the Basic Encoding Rules (BER) for ASN.1 which specifies how to encode ASN.1 data items into appropriate octet (8-bit byte) format. According to ASN.1, the value of any object is represented by three variable-length parts as shown in figure 3.4.

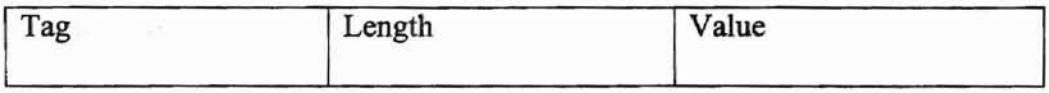

Figure 3.4 ASN.1 Object Structure

Here the tag field represents the type of the object, or type identifier, the length is the number of octets to hold the third part, the actual content of value. Figure 3.5 shows the common SNMPv1 type identifiers and their corresponding hex value.

Type	Hex	Type	Hex
INTEGER	02h	Time Ticks	43h
OCTET STRING	04h	Opaque	44h
NULL	05h	Get-Request PDU	a0h
OBJECT IDENTIFIER	06h	Get-Next-Request PDU	a1h
SEQUENCE, SEQ. OF	30h	Get-Response PDU	a2h
IPAddress	40h	Set-Request PDU	a3h
Counter	41h	Trap PDU	a4h
Gauge	42h	Get-Bulk-Request PDU	a5h

Figure 3.5 The Common SNMPv1 Type Identifiers

In particular, the type identifier of the SNMP data is SEQUENCE and has a value of 30h. An SNMP message consists of several data items and therefore is a complex nesting of encoded tag-length-value triplets.

In order to be able to represent arbitrarily large length value, SNMP uses the most significant bit of the length identifier octet as a flag to indicate if the length is encoded in a single octet. If this bit is zero, the short length encoding format is used and the least seven bits represent the value of the length (therefore short format only applies to the

cases where length is less than 128 octets). If the bit is a 1, the long format is used and the least seven bits represent the number of subsequent octets that hold the value of the length. The long format applies to arbitrary length value.

3.4 SNMP Protocol Format

According to the ASN.1, every SNMP message that is transmitted between an Manager and an agent has the following format:

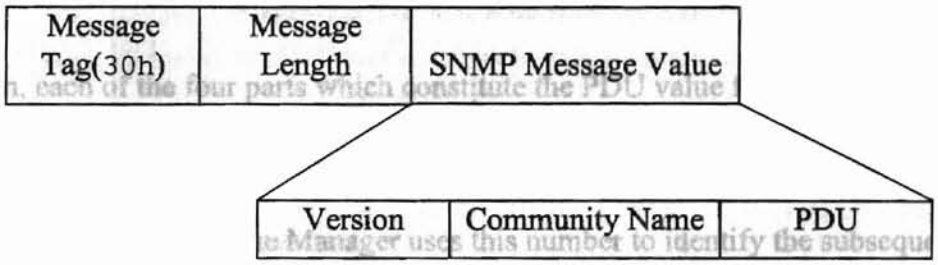

Figure 3.6 SNMP Message Format

Each of the three parts in the message value field must be represented by an ASN.1 triplet and be encoded according to the BER. For example, in the case of SNMP version1, the version field, with value zero, is encoded as follows (Note that the type of the *Version* object is INTEGER whose tag number is 2):

Version Tag(=2)	Version Length(1 octet)	Version Value (0)
0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0

Figure 3.7 An Example Of SNMP Message Field Encoding

The format of Protocol Data Unit (PDU) is

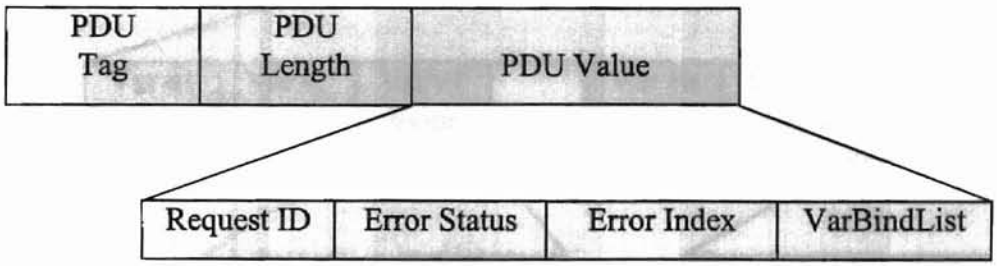

Figure 3.8 PDU Format

Again, each of the four parts which constitute the PDU value field is represented as an ASN.1 triplet. The request ID field is an "integer" that identifies the request sent from the Manager to the agent. The Manager uses this number to identify the subsequent matching response message from the agent. The error status field indicates whether there is an error and, in case of error(s), the type of the first error. The error index points to the first variable (object) in the variable binding list that caused the error. The variable binding list is a list of variables (objects) that a particular SNMP message carries. Its format is as follows:

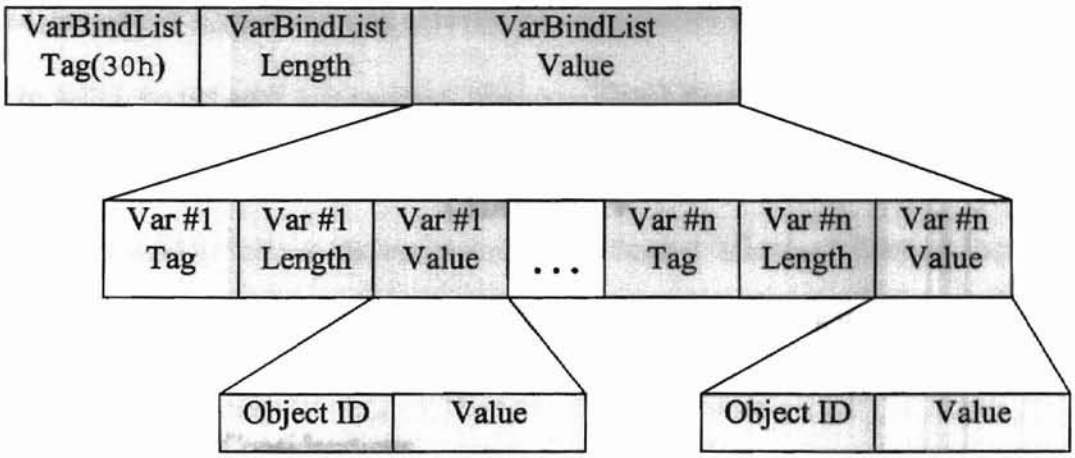

Figure 3.9 The Variable Binding List Format

where the object ID and the value field at the lowest level are further divided into their corresponding ASN.1 triplets.

CHAPTER IV

THE IMPLEMENTATION OF AN SNMP TOOL

4.1 Implementation Considerations

Microsoft Visual Basic 5.0 was chosen as the implementation language for all the related application development of this thesis. The primary reasons for choosing Visual Basic is that it offers a very easy way to create a graphic user interface. It is becoming more and more popular in business and education. Furthermore, if Visual Basic gains more capability in object oriented design and programming, it will be easy to enhance in the future to meet more complex network management requirements.

In order to conform to the SNMP rules to the fullest extent, one should not presume the size of any data type or data field involved in an SNMP message. For example, one can not assume an integer in SNMP has 16 or 32 bits. Every data field associated with an SNMP message must be packed according to the ASN.1 and be encoded according to the BER as described earlier. In Visual Basic, an appropriate data type to hold the SNMP packet is the byte array.

In order to take advantage of the object oriented features of Visual Basic, an ActiveX control ***snmpPacket***, which can be used to perform most of the SNMP property settings and packet encoding and decoding, was created using the ActiveX control development utilities that come with Visual Basic version 5. In the next chapter of this

thesis, the SNMP application (the network manager) which uses the ActiveX control as a tool to fulfill the network management task uses Visual Basic, too. Note that ActiveX controls written in Visual Basic look and behave just like controls written in C or C++. They can be used in various places: on web pages through Microsoft Internet Explorer, in Microsoft Visual C++, Visual Basic, Borland Delphi, and more.

4.2 AN SNMP Tool — snmpPacket Control

With Visual Basic 5, it is very easy to create and debug an ActiveX control. If a user chooses the "ActiveX control" option when creating a new project, Visual Basic automatically creates a template for drawing the interfaces and designing all the properties, methods, and events of the control. A project group can also be formed to include in it the ActiveX control project and a standard EXE project which allows the control to be tested and debugged in a very convenient way. For this thesis, a control called ***snmpPacket*** control was created whose primary responsibilities are encoding and decoding SNMP packets. The following properties are designed for the control with which the users of the control can interact to get or set values that are important in an SNMP message

1. `Version`. Type of byte, version of SNMP. The default value is 0 (version 1).
2. `CommName`. Type of string, the community name of the SNMP message. The default value is "public".
3. `PduType`. Type of byte, the SNMP command ID. The valid values are: GET-REQUEST (`160`), GET-NEXT-REQUEST (`161`), GET-RESPONSE (`162`), SET-REQUEST(`163`), and TRAP (`164`).

4. *ReqID*. Type of byte, request ID. The user of the control is responsible for setting or checking the request ID of an SNMP message. The default value is zero.

5. *ErrStatus*. Type of byte, error status. The recognized values are:

Status Name	Status Value
NoError	0
TooBig	1
NoSuchName	2
BadValue	3
ReadOnly	4
GenericError	5

6. *ErrIndex*. Type of byte, error index.

7. *Oid*. Type of string, object identifier.

8. *VarValType*. Type of *objectType* which is defined as an enumeration. The valid values are:

Type Name	Value	Type Name	Value
INTEGER_TAG	2	IP_ADDR_TAG	64
STRING_TAG	4	COUNTER_TAG	65
NULL_TAG	5	GAUGE_TAG	66
OID_TAG	6	TIME_TICK_TAG	67
UNIV_SEQ_TAG	48	OPAQUE_TAG	68

9. `VarVal`. Type of string, represents value of the variable.

The `Property Set` and `Property Get` statements were used in defining each of those properties. The value of each property is stored in a private module level variable. All those properties were defined as public properties. They serve as interfaces for a user to access the corresponding private variables.

An "*initialize()*" event has been designed so that the users of the control have a chance to initialize some basic properties of the control such as the "Version", "CommName" properties, and so on.

There are two methods implemented in the control: *packetize(ByRef ba() As Byte)*, which packs all the information set in the control into a byte array *ba()* that is ready to be sent between the manager and agent, and *unpack (ByRef ba() As Byte, ByRef resultVal As String),* which takes an SNMP packet (a byte array) *ba()* as input, unpacks it, checks for possible errors in the packet, and, if no errors occur, extracts the object identifier and the corresponding value. The extracted value is stored in the string *resultVal*.

Two module-level public enumeration data types have been defined: *SnmpType*, which represents the type of SNMP commands, and *objectType*, which represents the type of managed object or variable. Having these not only prevents the user of the control from messing up type of data but also makes those data types available to the Visual Basic object browser.

There are also some private sub procedures defined for the control which are not available to the user of the control. For example, the sub procedure *oidByte()* converts an object identifier string into a byte array that only holds the numbers in the object identifier. The sub procedure *stringByte()* converts a string or number (may be integer or long integer) into a byte array.

A user defined data type *mibNode* has also been defined which contains two strings, one is the OID of an object and another is a textual description of the object.

```
Type mibNode
   oid as string
   desc as string
End type
```

A MIB table was defined as an array of *mibNode* which is initialized when the control is loaded. This table is useful when the control unpacks an SNMP message.

Finally, the ***snmpPacket*** control, like the Timer control in Visual Basic, is only visible at design time.

CHAPTER V

AN SNMP APPLICATION — vbsnmp

In this application, a simple yet flexible and expandable network manager, **vbsmnp**, was created which can be used to fetch and set values of managed objects from a given SNMP agent. Users can use these values to analyze and manage network. Three operations are supported: GET-REQUEST. GET-NEXT-REQUEST, and SET-REQUEST.

5.1 Interfaces And Components

The application consists of two standard forms: *frmSnmp,* which is the main form of the application, and *frmSet,* which is used to set an SNMP value, and a module *mdlSnmp* in which constants, user-defined data types, public variables and procedures are defined.

The main form has the following components: a Microsoft Winsock control, the SnmpPacket control, a tree view control, three option buttons, two command buttons and several text boxes. Figure 5.1 shows a snapshot of the main window when the value of the object *sysUpTime*, which indicates how long the system has been running since its last start up, has been fetched from the SNMP agent installed in a server named "sro".

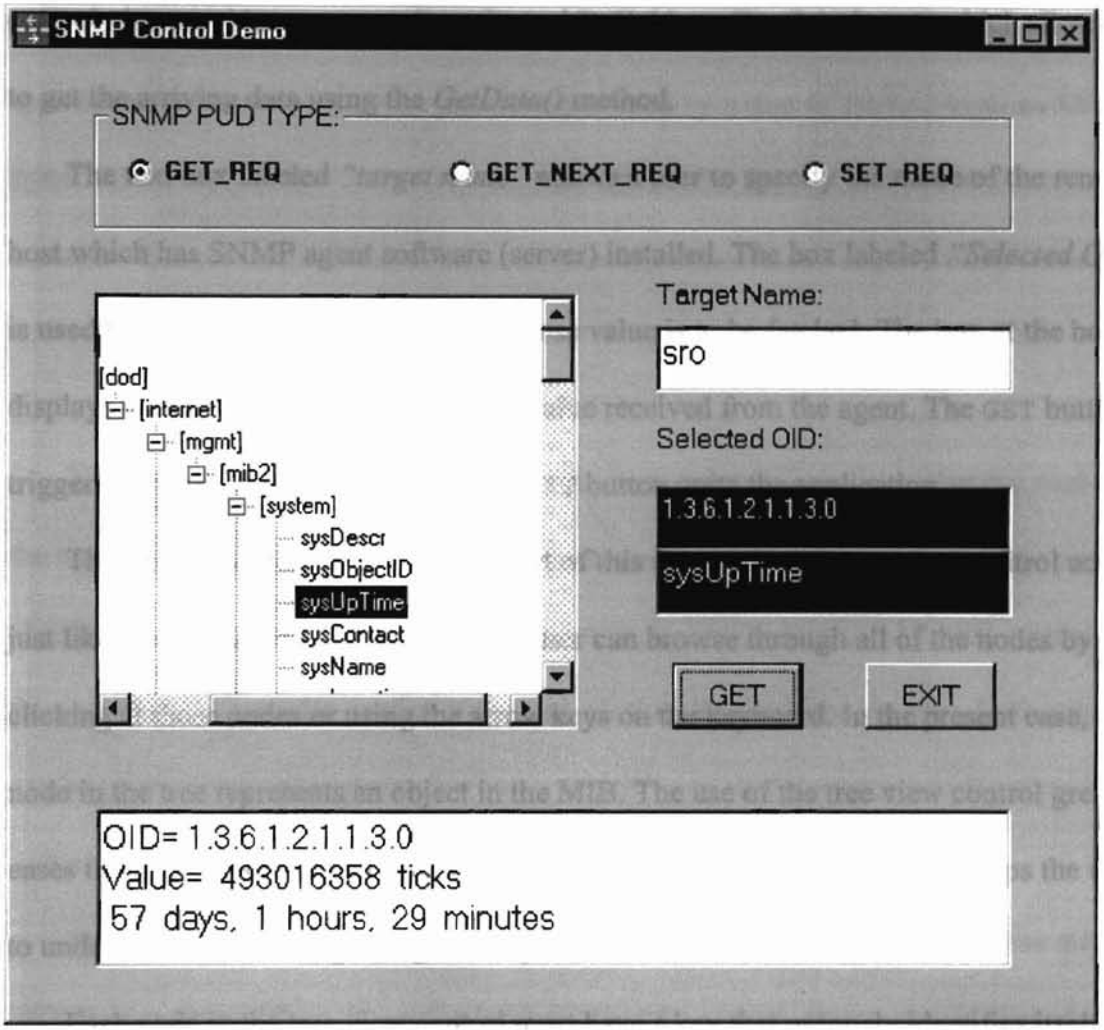

Figure 5.1 SNMP Control Demo

The SnmpPacket control and the Winsock control play major roles in the application. The former is used to set various values in an SNMP message, pack the message into a byte array before sending the packet to an SNMP agent, and unpack an incoming SNMP packet to extract the value of the inquired object. The latter is used to transmit packets between the SNMP manager (this application) and a remote SNMP agent. The method *sendData* sends a packet to the remote agent in the form of a byte array. The Winsock

control also provides an event, *DataArrival(ByVal bytesTotal As Long)*, which allows us to get the arriving data using the *GetData()* method.

The text box labeled *"target name"* allows a user to specify the name of the remote host which has SNMP agent software (server) installed. The box labeled *"Selected OID"* is used to specify the object identifier whose value is to be fetched. The box at the bottom displays the OID and the corresponding value received from the agent. The `GET` button triggers transmission of data while the `EXIT` button quits the application.

The tree view control is another effort of this application. A tree view control acts just like the Windows Explorer so that a user can browse through all of the nodes by clicking at those nodes or using the arrow keys on the keyboard. In the present case, each node in the tree represents an object in the MIB. The use of the tree view control greatly eases the user's task in specifying the object identifier of an object. It also helps the user to understand the meaning of a given object identifier.

Each node in the tree view control must have a key that uniquely identifies itself and could have optional text for its description. In the present case, the object identifier could have been used as the key since it is the unique name of an object in the MIB. However, a string of dot-separated numbers is not a valid key. Therefore the key that was actually used for each node in the tree view control is an object identifier prefixed by the letter "k". The corresponding textual name of the object was used as the optional description text for the node.

A dynamically allocated array, *mibTable()*, of the user-defined data type *mibNode* (see section 4.2) has been declared to hold a collection of object identifiers and their corresponding textual descriptions. The purpose of this array is two fold: first, it provides

a data source to load up the tree view control, and second, it serves as a database for the application to interpret an object identifier provided by a user or fetched from an SNMP agent. All the data items in the array are read from a text file "*mib.txt*" in which each line contains an object identifier and the corresponding textual name (separated by a comma and zero or more spaces). The following line is an example entry in the text file:

```
1.3.6.1.2.1.1.3.0 , sysUpTime
```

The size (bound) of the array depends on the number of data items that are read from the text file. This makes the application very expandable: if one wants to include more objects in the application, the only thing he or she needs to do is to add more entries to the text file. Furthermore, entries in the text file do not have to be in order. In fact they can be in any order. The application automatically puts all the entries into the correct nodes in the MIB tree.

The three option buttons are used to allow users of the application to choose different SNMP operations. When the option button `GET-REQ` or `GET-NEXT-REQ` is clicked, the *pduType* property of the *SnmpPacket* control is set to its corresponding value and the *varValType* property is set to `NULL`. While when the option button `SET-REQ` is clicked, a new window, as shown in figure 5.2, is popped out to prompt the user to input setting values such as the OID of an object and the corresponding value. Since the window is popped out as a modal window, the user must first respond to this window, either by filling out all the required fields and clicking the **OK** button, or by clicking the **CANCEL** button.

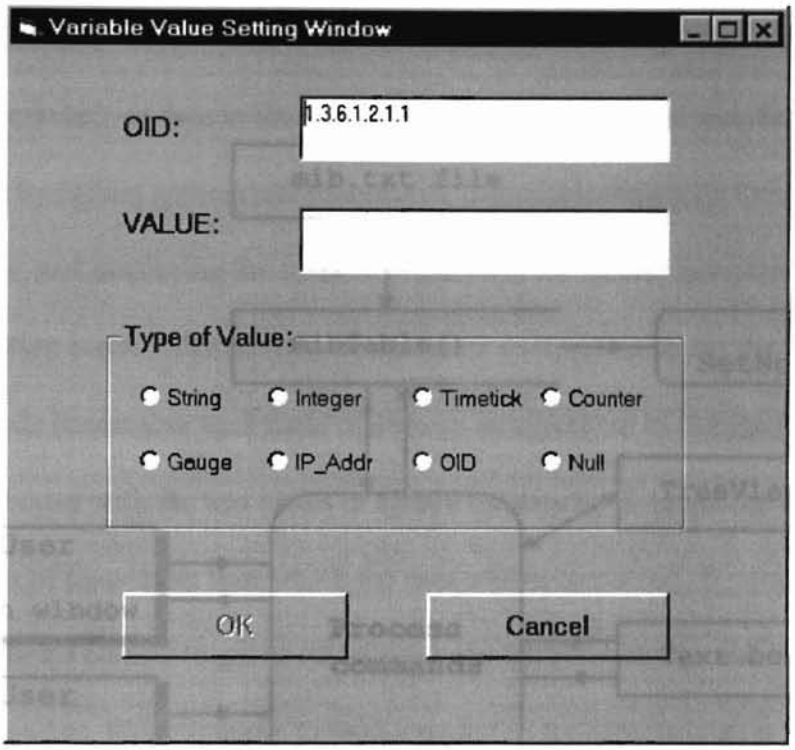

Figure 5.2 Variable Value Setting Window

5.2 Data Flows And Algorithms

Figure 5.3 shows the data flows of the application. The shadowed rectangles represent the user interfaces with which users can interact with the application. In this case they are the main application window and the window for setting a value of an OID. The open ended rectangles are stores of data or controls that contain both stores of data and methods. The rounded rectangles represent processes that consist of one or more sub procedures. The arrows represent the directions of data flows.

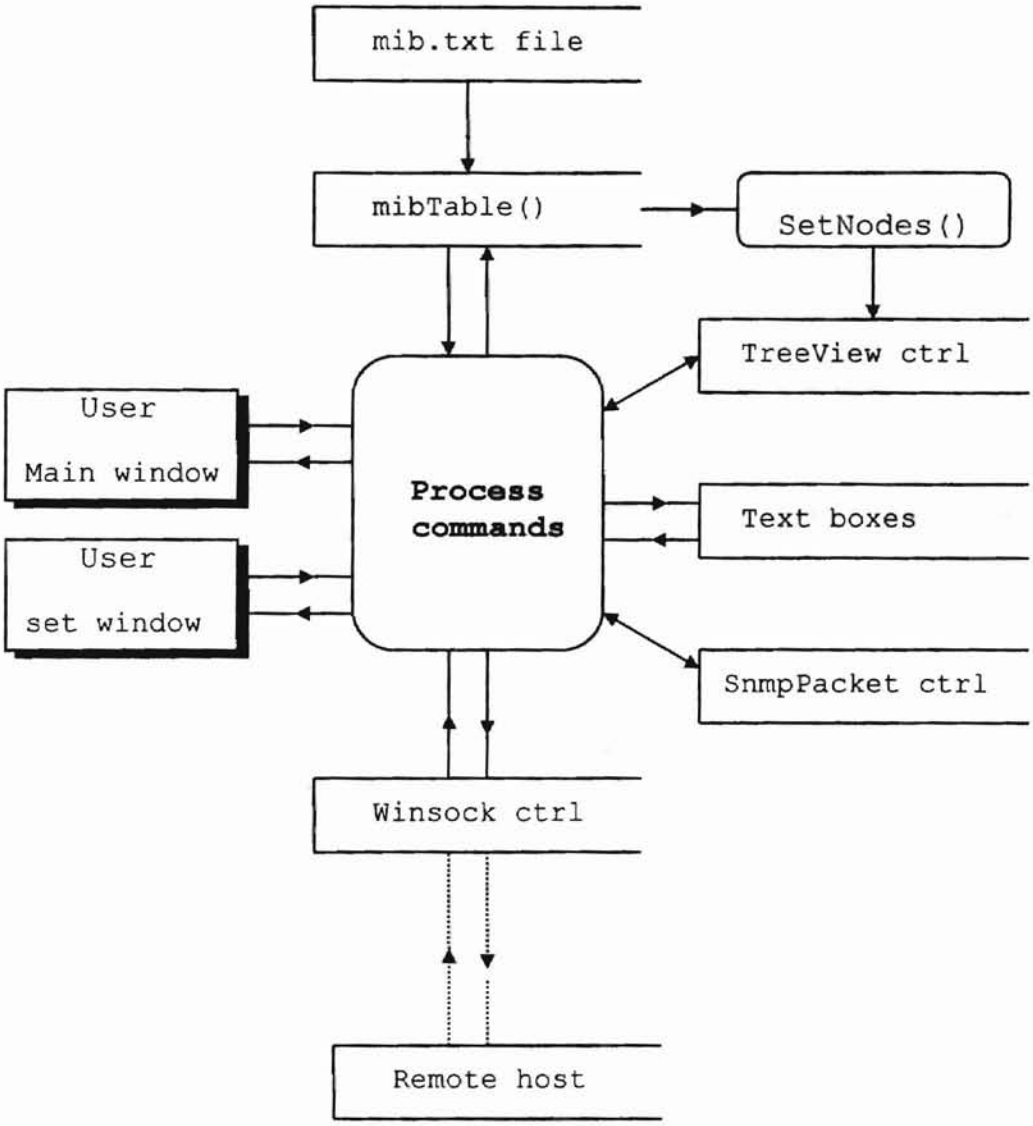

Figure 5.3 Data Flow Diagram

When the application starts, it first reads all the available OIDs and corresponding textual names into the array *mibTable()* from the text file *mib.txt*. A subroutine *setNodes()* is then called to load these data to the tree view control so that all the nodes in the tree are positioned correctly. Next, the application does some necessary initializations to the *SnmpPacket* control, *Winsock* control, and the text boxes, and finally displays the main

window. The user can now interact with the application through the interface(s) by sending commands or data to the application. The application responds to these commands by calling appropriate procedures, communicating with the stores of data and the controls, and displaying the updated window(s) for further interaction. Specifically, the application communicates with the tree view control to retrieve the information of the selected node (remember each node represents an object) or to change the selected node. It communicates with the text boxes to update the display or to get the displayed text such as the name of the remote host which the user wishes to contact. It communicates with the *SnmpPacket* control to get or set values in an SNMP message and to pack or unpack an SNMP packet. When it needs to send a packet to the remote host, it communicates with the *Winsock* control.

A subprocedure *readMib()* was used to read data items from the text file "*mib.txt*". In the procedure, the following statement was used to open the file:

```
Open App.Path & "\mib.txt" For Input As #1
```

where App is a global object provided by Visual Basic for determining or specifying properties of the application. `App.Path` is the path of the `.exe` file of the application. The statement "`Line Input #1,v`" was used inside a loop to read one line from the file each time and store it in a string `v`.

The procedure then loads the data items contained in "*mib.txt*" into a temporary Visual Basic *collection* object `mibColl` which was declared as:

```
Public mibColl As New Collection
```

A binary search algorithm was used to insert the data items into the collection so that all of the data items were sorted according to their OIDs. These sorted data items in the

collection are then moved to the *mibTable()*, with the object whose OID has smallest string value being moved first. Next, a procedure *setNodes()* is called to load the data items onto the tree view control. The following statement was used to add a node to the tree

```
Set nodX = tvwOid.Nodes.Add(relative, tvwChild, _
                    "k" & mibTable(i).oid, mibTable(i).desc)
```

where `tvwoid` is the name of the tree view control, `Nodes` is a `collection` object which contains a collection of `Node` objects, `"k"& mibTable(i).oid` is the key for the node to be added, `mibTable(i).desc` is the text of the node, and `relative` is the parent node which determines the position of the current node in the tree. In order to position a node correctly in the tree, one must first find its parent node. The criteria for a node (A) to be the parent node of a node (B) are: 1), the OID of node A is a left substring of the OID of node B and 2), the number of dots in the OID of node A is at least one less than that in the OID of node B and the difference should be minimum. Sorting data items in *mibTable()* is very important in ensuring that the whole tree is structured correctly.

CHAPTER VI

EXAMPLES OF THE USE OF THE APPLICATION

To demonstrate the use of this network management application, let's take the objects in the ip group in the MIB tree as examples.

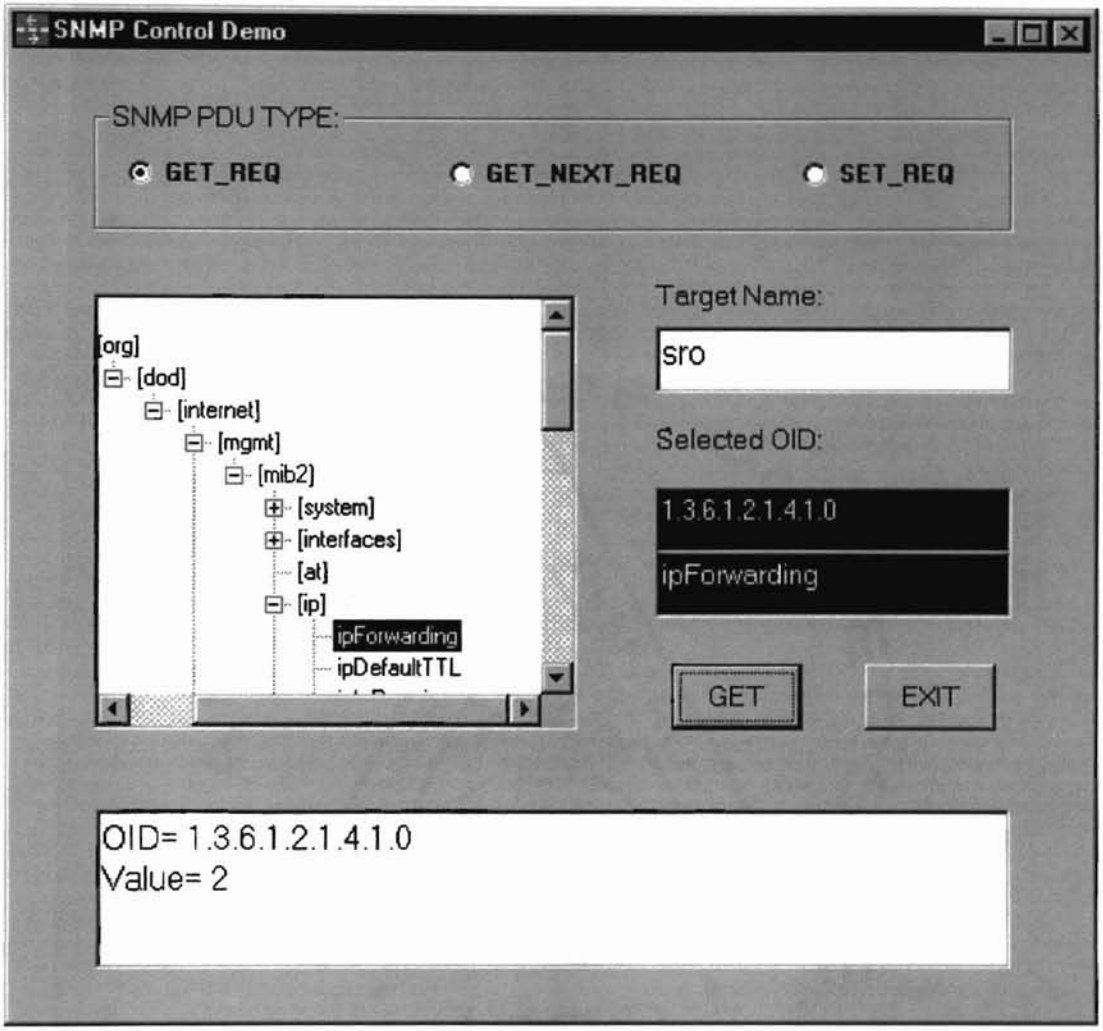

Figure 6.1 The Result of ipForwarding Object

Figure 6.1 shows the result by querying the ipForwarding object on server "sro".

According to SNMP protocol specifications, ipForwarding returns a integer, 1 means the system is forwarding IP datagrams, and 2 means it is not. The figure shows that "sro" is not a IP gateway which forwarding datagrams.

Figure 6.2 shows the result by querying the ipDefaultTTL object on server "sro".

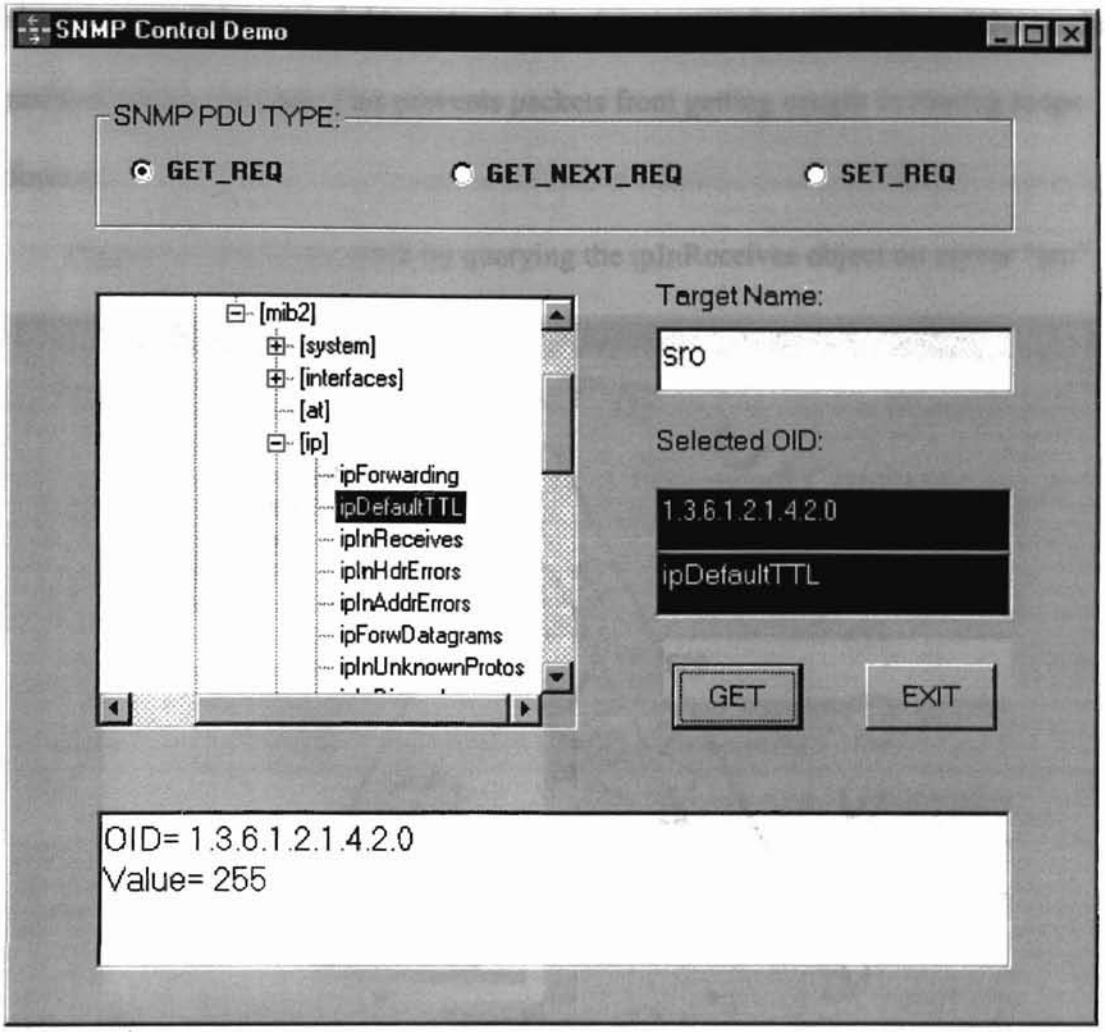

Figure 6.2 The Result of ipDefaultTTL Object

The ipDefaultTTL object represents the default value inserted into the Time-To-Live field of the IP header of datagrams originated at this entity, whenever a TTL value is not supplied by the transport layer protocol. The figure shows that the TTL value is 255 seconds/hops. This IP header field sets an upper limit on the number of routers through which a datagram can pass. It limits the lifetime of the datagram. It is initialized by the sender to some value (often 32 or 64) and decreased by one by every router that handles

the datagram. When this field reaches 0 , the datagram is thrown away, and the sender is notified with a message. This prevents packets from getting caught in routing loops forever.

Figure 6.3 shows the result by querying the ipInReceives object on server "sro".

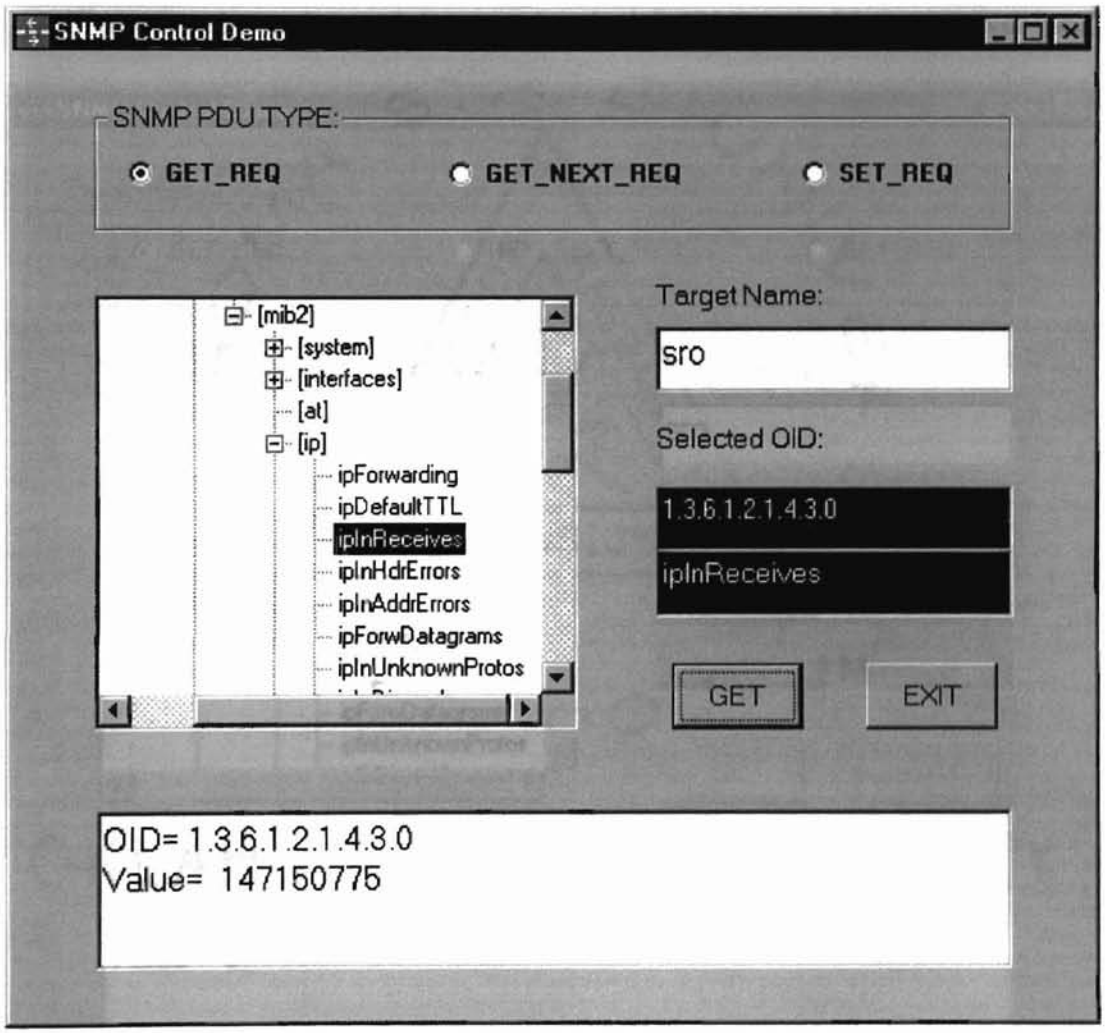

Figure 6.3 The Result of ipInReceives

The ipInReceives object represents the total number of input datagrams received from interfaces, including those received in error. By querying this kind of numbers, the network administrators can figure out the traffic situation on a specific network node, this information can then be used to the design and upgrade of the network.

Figure 6.4 shows the result by querying the ipInHdrErrors object on server "sro".

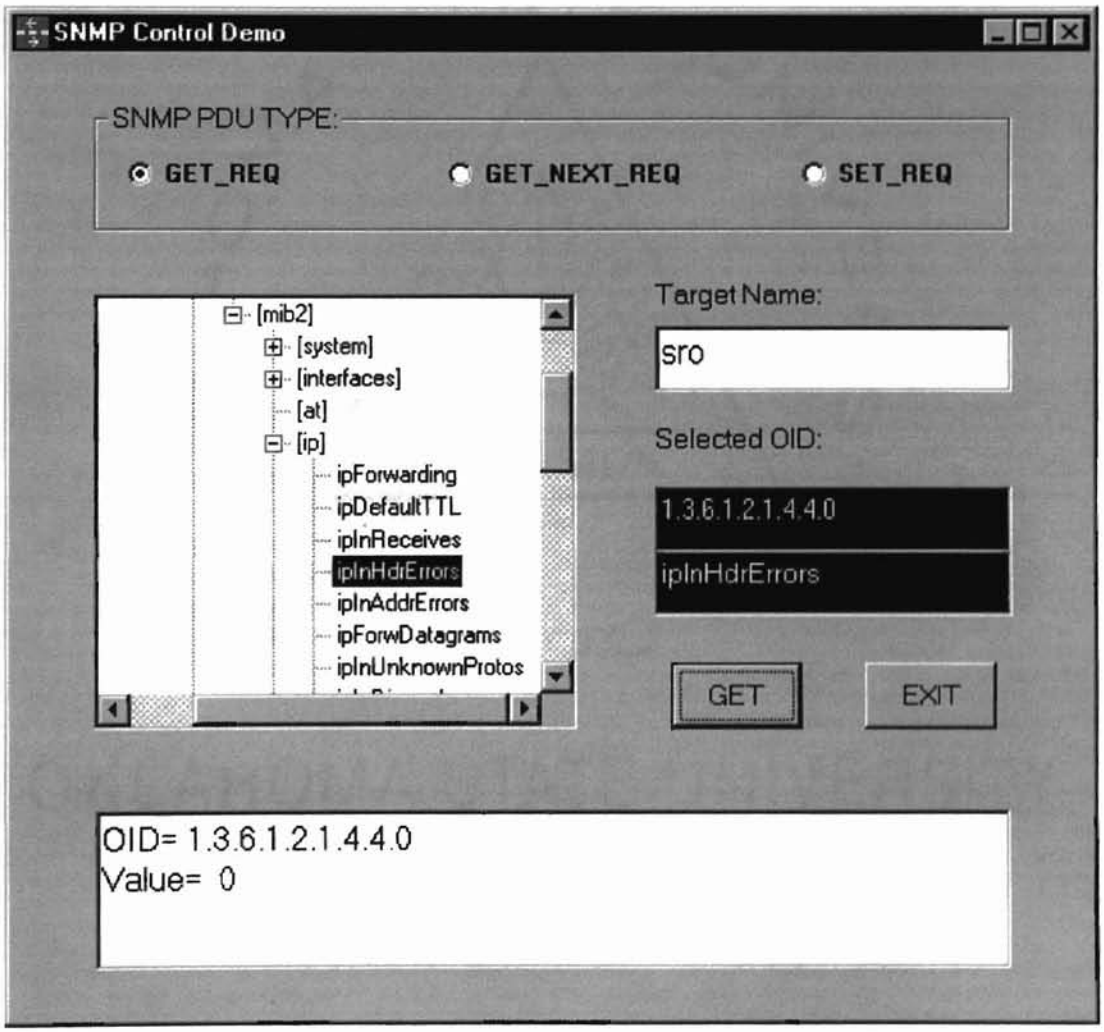

Figure 6.4 The Result of ipInHdrErrors Object

The ipInHdrErrors is a counter, it represents the number of input datagrams discarded due to errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options, etc. The figure shows there is no datagrams discarded.

Figure 6.5 shows the result by querying the ipInDelivers object on server "sro".

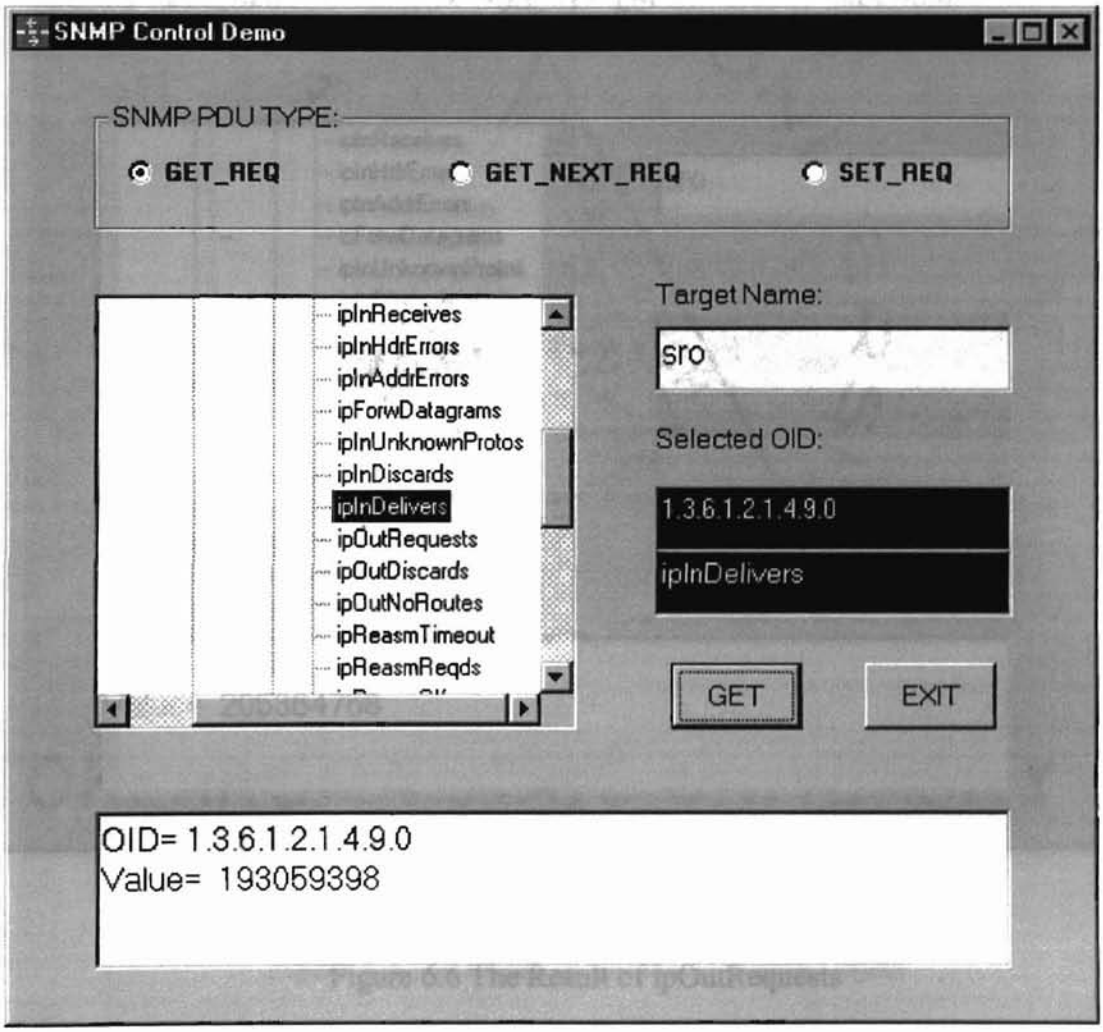

Figure 6.5 The Result of ipInDelivers Object

The ipInDelivers object is a counter, it represents the total number of input datagrams successfully delivered to IP user-protocols.

Figure 6.6 shows the result by querying the ipOutRequests object on server "sro".

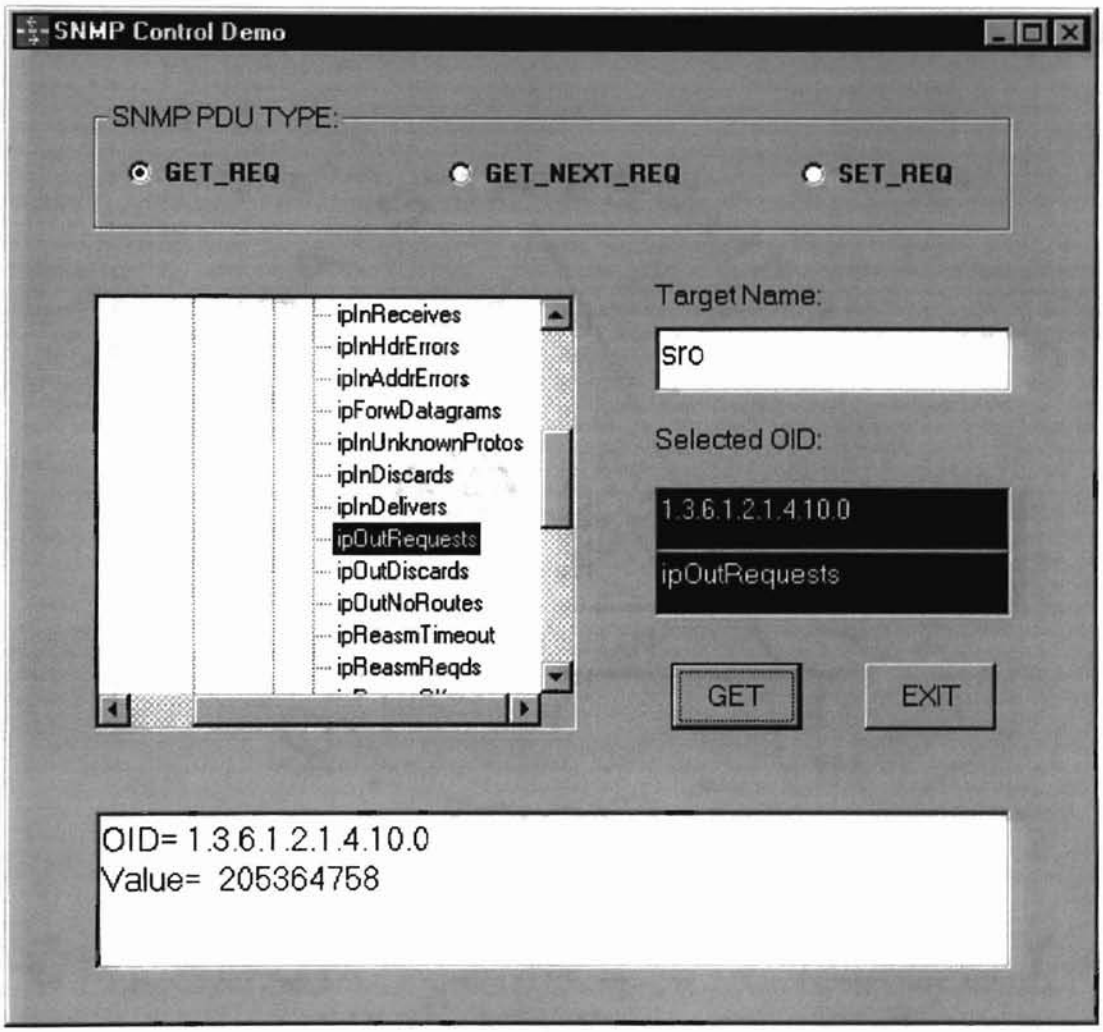

Figure 6.6 The Result of ipOutRequests

The ipOutRequest object represents the total number of IP datagrams which local IP user-protocols supplied to IP in requests for transmission. This counter does not include any datagrams counted in ipForwDatagrams.

Figure 6.7 shows the result by querying the ipReasmTimeout object on server "sro".

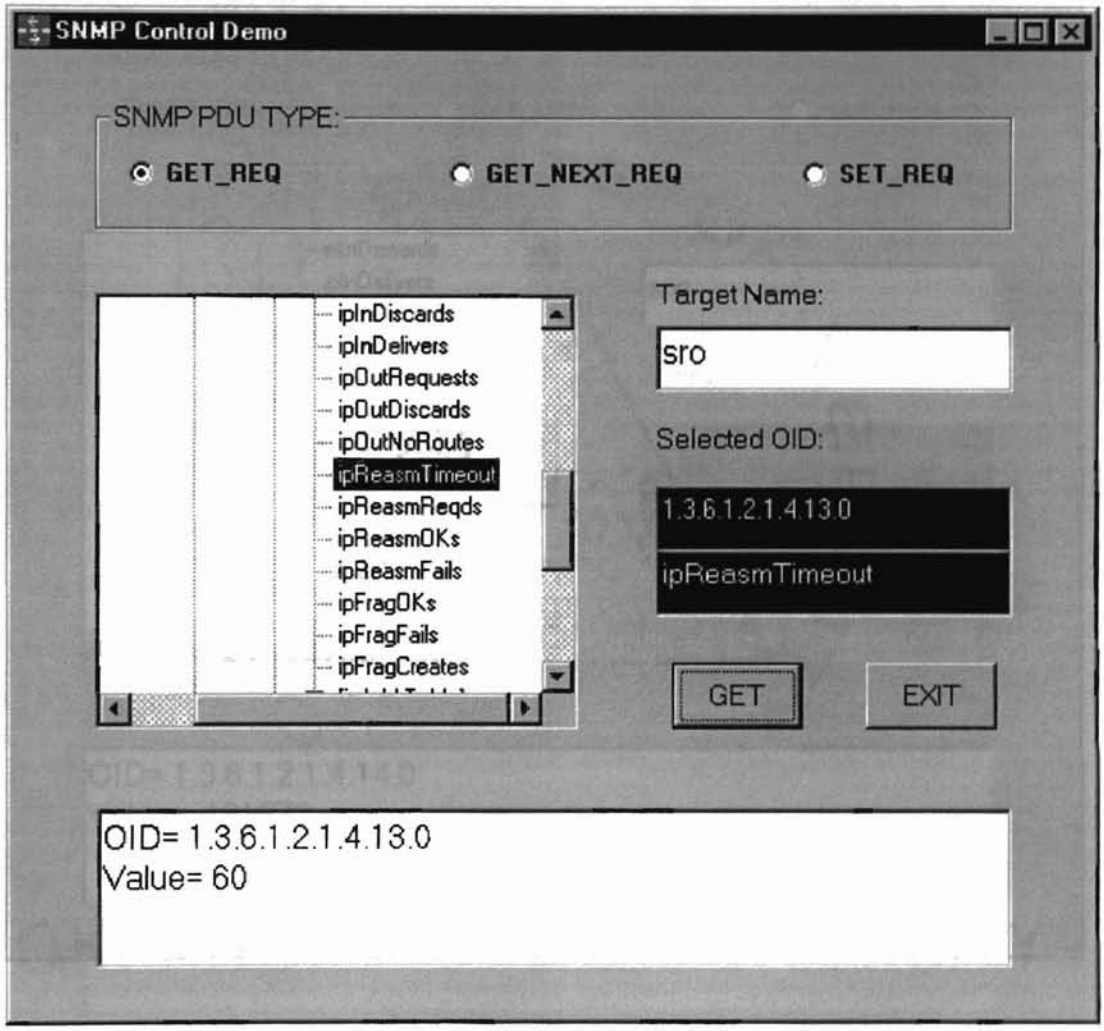

Figure 6.7 The Result of ipReasmTimeout Object

The ipReasmTimeout object represents the maximum number of seconds which received fragments are held while they are awaiting reassembly at this entity. The figure shows the number is 60 seconds.

Figure 6.8 shows the result by querying the ipReasmReqds object on server "sro".

The ipReasmReqds object is a counter, it records the number of IP fragments received which needed to be reassembled at this entity.

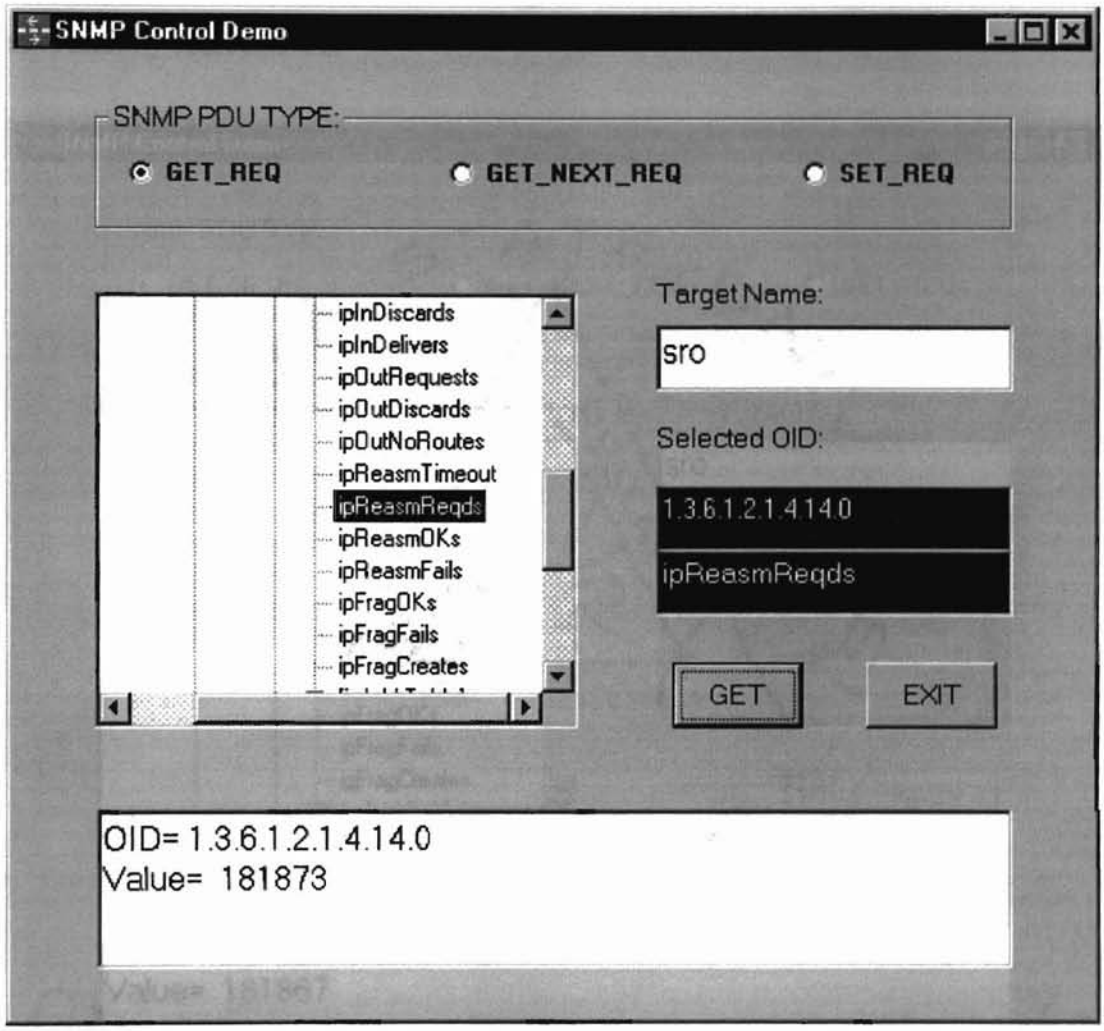

Figure 6.8 The Result of ipReasmReqds Object

Figure 6.9 shows the result by querying the ipReasmOKs object on server "sro".

The ipReasmOKs object is a counter, it records the number of IP datagrams successfully reassembled.

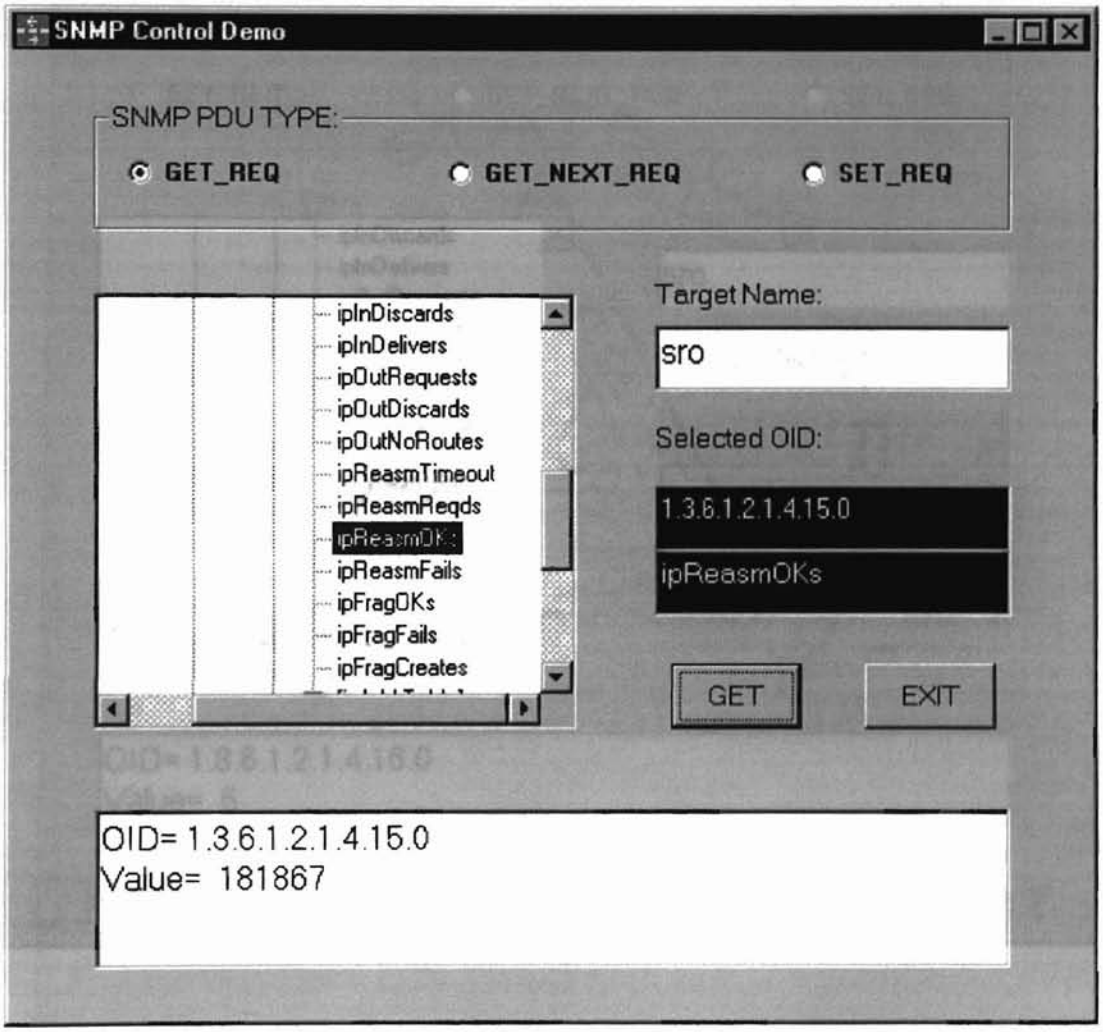

Figure 6.9 The Result of ipReasmOKs Object

Figure 6.10 shows the result by querying the ipReasmFails object on server "sro".

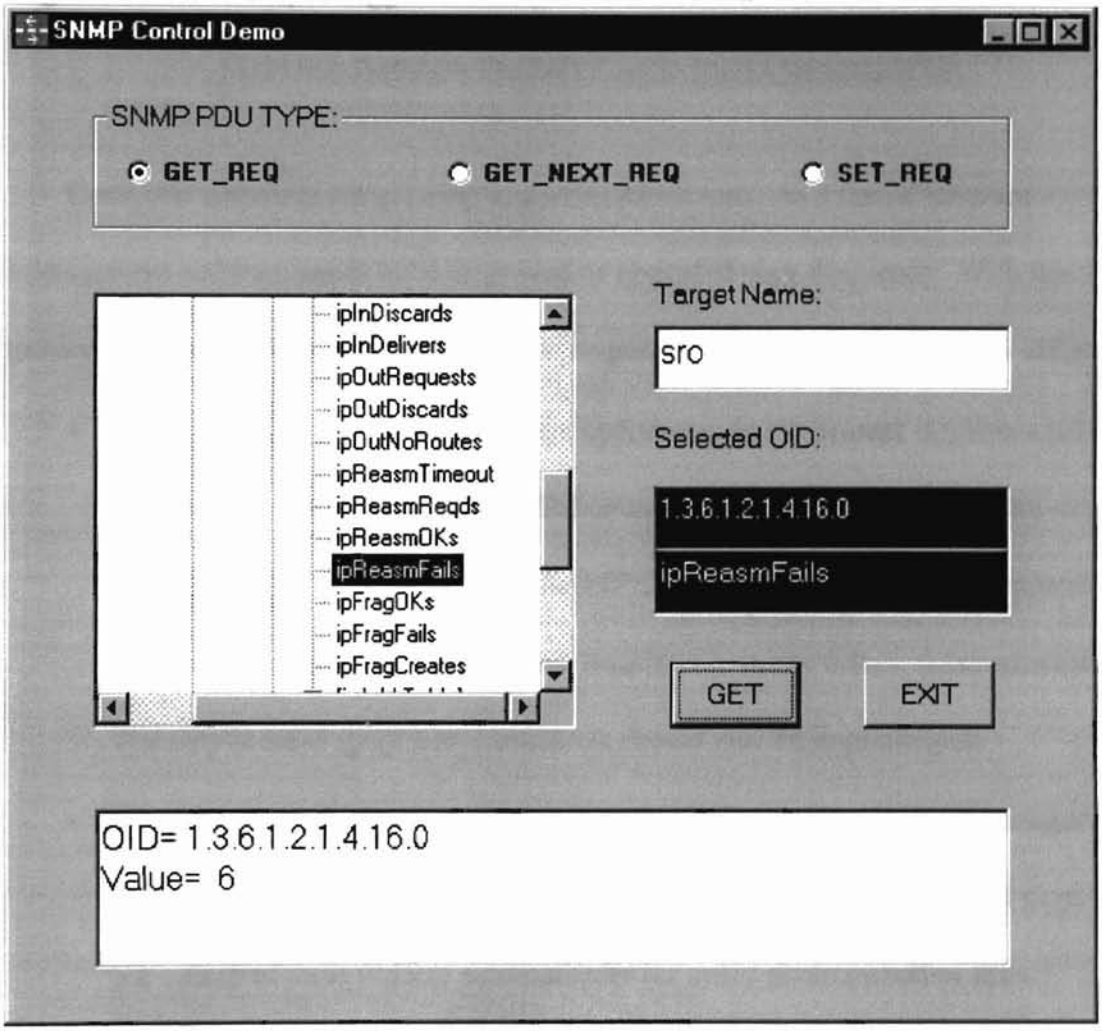

Figure 6.10 The Result of ipReasmFails Object

The ipReasmFails object is a counter, it records the number of failures detected By the IP reassembly algorithm (for what ever reason: timed out, errors, etc). Note that this is not necessarily a count of discarded IP fragments since some algorithm can lose track of the number of fragments by combining them as they are received. Figure 6.8

shows 181,873 fragments received, Figure 6.9 shows 181867 fragments successfully reassembled, Figure 6.10 shows there are 6 fragments failed, sometimes this failed fragments number maybe bigger due to the reason stated above.

CHAPTER VII

FUTURE IMPROVEMENTS AND ENHANCEMENTS

Computer networks are growing at a tremendous rate. As a result, network management software needs to be improved or upgraded very frequently. With this initial ground work, the following improvements are possible and should not be very difficult.

- Support more network management operations. In the present implementation, the manager only supports three SNMP operations: `GET-REQUEST`, `GET-NEXT-REQUEST`, `SET-REQUEST`. The SNMPv2 also defined two more commands: `GET-BULK-REQUEST` and `INFORM-REQUEST`. In the future, these commands and maybe some more new commands should also be implemented.
- Automated monitoring. In this part, a user can choose one or more managed objects which he/she wants to monitor and the application fetches and displays the values of these objects automatically for every given period of time.
- Statistics and better GUIs. The application could be improved so that it can automatically fetch data at different times and use these data to create some statistics. Charts and curves can also be used for better representation of the data.
- Learning ability. By using the `GET-NEXT-REQUEST` operation, a manager can get the knowledge of all the objects managed by a given SNMP agent. The manager can remember the set of the managed objects and can update the data periodically at a given period of time.

BIBLIOGRAPHY

[1] Case, J.D.; Fedor, M.; Schoffstall, M.L.; Davin, J.R. "A Simple Network Management Protocol." *RFC 1067*, August 1988, Internet Authority Board: http://www.faqs.org/rfcs/rfc1067.html

[2] McCloghrie, K.; Rose, M. " Structure and Identification of Management Information for TCP/IP-based Internets." *RFC 1065*, August 1988, Internet Authority Board: http://www.faqs.org/rfcs/rfc1065.html

[3] McCloghrie, K.; Rose, M., "Management Information Base for Network Management of TCP/IP-based Internets." *RFC 1066*, August 1988, Internet Authority Board: http://www.faqs.org/rfcs/rfc1066.html

[4] Case, J.D., etc. "Simple Network Management Protocol (SNMP)." *RFC 1098*, April, 1989, Internet Authority Board: http://www.faqs.org/rfcs/rfc1098.html

[5] Rose, M.; McCloghrie, K.; Davin, J.R.. " Bulk Table Retrieval with SNMP." *RFC 1187*, October 1990, Internet Authority Board: http://www.faqs.org/rfcs/rfc1187.html

[6] Harnedy, Sean. *Total SNMP: Exploring the Simple Network Management Protocol.* Second edition. Prentice Hall, Upper Saddle River, 1997.

[7] Stalling, William "SNMPv3: A Security Enhancement for SNMP. " *IEEE Communication Surveys*, 4th quarter, 1998.

[8] Stalling, William "Security Comes to SNMP: The New SNMPv3 Proposed Internet Standards." *The Internet Protocol Journal*, Dec. 1998.

[9] Stamatelopoulos, F.; Maglaris, B. " A Management Architecture for Internet Information Services." Presented at the *HP OpenView University Association Workshop*, Madrid, April 1997.

[10] Deri, Luca; etc. "JLOCATOR: A Web-Based Asset Location System." Presented at the *HP OpenView University Association Workshop*, Madrid, April 1997.

[11] Steedman, Douglas. *Abstract Syntax Notation One (ASN.1): The Tutorial and Reference*. London, Techmology Appraisals, Ltd. 1990.

[12] Rose, M. *The Simple Book An Introduction to Management of TCP/IP-based Networks*. Prentice Hall, Upper Saddle River, New Jersey, 1990.

[13] Perkins, David; McGinnis, Evan. *Understanding SNMP MIBs*. Prentice Hall, Upper Saddle River, New Jersey, 1997.

[14] McCloghrie, K. "Management Information Base for Network Management of TCP/IP-based Internets." *RFC 1156*. May 1990. Internet Authority Board: http://www.faqs.org/rfcs/rfc1156.html

[15] McCloghrie, K.; Rose, M. " Management Information Base for Network Management of TCP/IP-based Internets: MIB-II." *RFC 1213*, March 1991. Internet Authority Board: http://www.faqs.org/rfcs/rfc1213.html

[16] Perkins, David. " How to Read and Use an SNMP MIB." 3TECH, *The 3Com Technical Journal*(Spring 1991): 31-55.

[17] Stewart, Bob. " Development an Integration of a Management Information Base." *ConneXions* (June 1991): 2-11.

[18] Anonymous, " Introduction to Simple Network Management Protocol: A Self-Study Guide." Document 8759-00, rev. A, 3Com Corporation, August 1991.

[19] Waldbusser, S. "Remote Network Monitoring Management Information Base." *RFC 1757*, February 1995. Internet Authority Board: http://www.faqs.org/rfcs/rfc1757.html

[20] Waldbusser, S. " Token Ring Extensions to the Remote Network Monitoring MIB." *RFC 1513*, September 1993. Internet Authority Board: http://www.faqs.org/rfcs/rfc1513.html

[21] Tolly, Kevin; Carr, Jim " RMON: A Ray of Hope for Token Ring Managers." *Data Communications* (October 1995): 72-80.

[22] Rose, M.; McCloghrie, K. "Structure and Identification of Management Information for TCP/IP-Based Internets." RFC 1155, May 1990. Internet Authority Board: http://www.faqs.org/rfcs/rfc1155.html

APPENDIX 1

THE MIB.TXT FILE

Note: the lines can be in any order, and more lines (objects) can be added if the agents in the network nodes support more MIB objects.

```
1  , [iso]
1.3  , [org]
1.3.6  , [dod]
1.3.6.1  , [internet]
1.3.6.1.2  , [mgmt]
1.3.6.1.2.1  , [mib2]
1.3.6.1.2.1.1  , [system]
1.3.6.1.2.1.1.1.0  , sysDescr
1.3.6.1.2.1.1.2.0  , sysObjectID
1.3.6.1.2.1.1.3.0  , sysUpTime
1.3.6.1.2.1.1.4.0  , sysContact
1.3.6.1.2.1.1.5.0  , sysName
1.3.6.1.2.1.1.6.0  , sysLocation
1.3.6.1.2.1.1.7.0  , sysServices
1.3.6.1.2.1.2  , [interfaces]
1.3.6.1.2.1.2.1.0  , ifNumber
1.3.6.1.2.1.3  , [at]
1.3.6.1.2.1.4  , [ip]
1.3.6.1.2.1.4.1.0  , ipForwarding
1.3.6.1.2.1.4.2.0  , ipDefaultTTL
1.3.6.1.2.1.4.3.0  , ipInReceives
1.3.6.1.2.1.4.4.0  , ipInHdrErrors
1.3.6.1.2.1.4.5.0  , ipInAddrErrors
1.3.6.1.2.1.4.6.0  , ipForwDatagrams
1.3.6.1.2.1.4.7.0  , ipInUnknownProtos
1.3.6.1.2.1.4.8.0  , ipInDiscards
1.3.6.1.2.1.4.9.0  , ipInDelivers
1.3.6.1.2.1.4.10.0  , ipOutRequests
1.3.6.1.2.1.4.11.0  , ipOutDiscards
1.3.6.1.2.1.4.12.0  , ipOutNoRoutes
1.3.6.1.2.1.4.13.0  , ipReasmTimeout
1.3.6.1.2.1.4.14.0  , ipReasmReqds
1.3.6.1.2.1.4.15.0  , ipReasmOKs
1.3.6.1.2.1.4.16.0  , ipReasmFails
1.3.6.1.2.1.4.17.0  , ipFragOKs
1.3.6.1.2.1.4.18.0  , ipFragFails
1.3.6.1.2.1.4.19.0  , ipFragCreates
1.3.6.1.2.1.4.20  , [ipAddrTable]
1.3.6.1.2.1.4.20.1  , [ipAddrEntry]
1.3.6.1.2.1.4.20.1.1  , [ipAdEntAddr]
```

```
1.3.6.1.2.1.4.20.1.2  , [ipAdEntIfIndex]
1.3.6.1.2.1.4.20.1.3  , [ipAdEntNetMask]
1.3.6.1.2.1.4.20.1.4  , [ipAdEntBcastAddr]
1.3.6.1.2.1.4.20.1.5  , [ipAdEntReasmMaxSize]
1.3.6.1.2.1.4.22  , [ipNetToMediaTable]
1.3.6.1.2.1.4.22.1  , [ipNetToMediaEntry]
1.3.6.1.2.1.4.22.1.1  , [ipNetToMediaIfIndex]
1.3.6.1.2.1.4.22.1.2  , [ipNetToMediaPhysAddr]
1.3.6.1.2.1.4.22.1.3  , [ipNetToMediaNetAddr]
1.3.6.1.2.1.4.22.1.4  , [ipNetToMediaType]
1.3.6.1.2.1.4.23.0  , ipRoutingDiscards
1.3.6.1.2.1.5  , [icmp]
1.3.6.1.2.1.6  , [tcp]
1.3.6.1.2.1.7  , [udp]
1.3.6.1.2.1.8  , [egp]
1.3.6.1.2.1.10  , [transmission]
1.3.6.1.2.1.11  , [snmp]
1.3.6.1.2.1.11.1.0  , snmpInPkts
1.3.6.1.2.1.11.2.0  , snmpOutPkts
1.3.6.1.2.1.11.3.0  , snmpInBadversions
1.3.6.1.2.1.11.4.0  , snmpInBadCommunityNames
1.3.6.1.2.1.11.5.0  , snmpInBadCommunityUsers
1.3.6.1.2.1.11.6.0  , snmpInASNParseErrs
1.3.6.1.2.1.11.8.0  , snmpInTooBig
1.3.6.1.2.1.11.9.0  , snmpInNoSuchNames
1.3.6.1.2.1.11.10.0  , snmpInBadValues
1.3.6.1.2.1.11.11.0  , snmpInReadOnlys
1.3.6.1.2.1.11.12.0  , snmpInGenErrs
1.3.6.1.2.1.11.13.0  , snmpInTotalReqVars
1.3.6.1.2.1.11.14.0  , snmpTotalSetVars
1.3.6.1.2.1.11.15.0  , snmpInGetRequests
1.3.6.1.2.1.11.16.0  , snmpInGetNexts
1.3.6.1.2.1.11.17.0  , snmpInSetRequests
1.3.6.1.2.1.11.18.0  , snmpInGetResponses
1.3.6.1.2.1.11.19.0  , snmpInTraps
1.3.6.1.2.1.11.20.0  , snmpOutTooBigs
1.3.6.1.2.1.11.21.0  , snmpOutNoSuchNames
1.3.6.1.2.1.11.22.0  , snmpOutBadValues
1.3.6.1.2.1.11.24.0  , snmpOutGenErrs
1.3.6.1.2.1.11.25.0  , snmpOutGetRequests
1.3.6.1.2.1.11.26.0  , snmpOutGetNexts
1.3.6.1.2.1.11.27.0  , snmpOutSetRequests
1.3.6.1.2.1.11.28.0  , snmpOutgetResponses
1.3.6.1.2.1.11.29.0  , snmpOutTraps
1.3.6.1.2.1.11.30.0  , snmpEnableAuthenTraps
1.3.6.1.3  , [experimental]
1.3.6.1.4  , [private]
1.3.6.1.4.1  , [enterprise]
```

APPENDIX 2

APPLICATION OPERATING PROCEDURE

1. Hardware and Software Requirements

   Hardware: CPU with speed of 133 MHZ or above
   CPU with Random Access Memory of 16 MB or above
   CPU with spare disk space of at least 80 MB
   SVGA color monitor
   IBM compatible keyboard and mouse

   Software: Windows 95/98/NT operating systems
   Microsoft Visual Basic 5.0 or above compiler

2. Installation and Operating Procedure

   A. Create a file folder in Windows Explorer of your workstation.
   B. Copy all the files in the floppy disk to the folder you just created.
   C. Open the folder and find a file named vbsnmp.vbg.
   D. Double click on this file and the Microsoft Visual Basic project window will be opened.
   E. Run the application by click on the "▶" icon on the view bar.
   F. The SNMP Control Demo Window will pop out.
   G. Choose any MIB network object you are interested in from the left hand tree view control, type the network node name in the "Target Name" box, choose one of the three "SNMP PDU TYPE" radio button, then press the "GET" button, the selected MIB object value on the targeted network node will show on the bottom text box. (Note: if you have the authentication to set a variable value on a network node, then you can use the "SET_REQ" button to change the value of the network node variable.)

VITA

Wenxia Zhang

Candidate for the Degree of

Master of Science

Thesis: THE DESIGN AND IMPLEMENTATION OF A NETWORK MANAGEMENT APPLICATION USING SNMP PROTOCOL

Major Field: Computer Science

Biographical:

Personal Data: Born in Yangquan, Shaxi Province, China, June 7, 1966, the daughter of Xuezhou Zhang and Shizheng Li.

Education: Graduated in July, 1984 from Yangquan 1st High School in Shaxi, China. Received Bachelor of Science degree in Accounting from Tianjin Finance & Monetary University, China in July, 1994. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University, Stillwater in May, 2000.

Experience: Engineer and manager in the Finance and Accounting Information Systems Center of Tianjin University, Tianjin, China, 1988 – 1994.