DISTRIBUTED REPORTING

MANAGEMENT SYSTEM

By

CHRISTINE E. VALLIERE

Bachelor of Science

Eastern New Mexico University

Portales, New Mexico

1991

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 1996

DISTRIBUTED REPORTING

MANAGEMENT SYSTEM

Thesis Approved:

Mitchell A. Neilsen

Thesis Adviser

M. Samadzadeh

Blayne E. Mayfield

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

My thesis committee has been terrific in ensuring my thesis was completed in a timely manner. I would especially like to thank my advisor Dr. Mitch Neilsen for his guidance and advice throughout my research work. I would like to thank Dr. Mansur Samadzadeh for his support and his thorough analysis of my paper. And I would like to thank Dr. Blayne Mayfield for his guidance in the completion of this research.

My special appreciation to my husband, David, for his encouragement and support as I completed my graduate studies. Additionally, I would have never made it through the last three years without the patience, friendship, and assistance of Lee Fields, Greg Gudenburr, and Jay Simpson. Finally, it is my family and friends who have had to put up with me. Thanks to you all.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Performing network management over a distributed environment leads to many complexities. Typically Network Management functions are done on a server; however the server can become a bottleneck. Suppose there was a method that allowed the raw data, used to perform network management, to be processed on the client. This would reduce the load on the server. Additionally, if the server was able to provide the information on how to process the data, then the processing code would only have to be stored (and changed) on the server. The new language from *Sun Microsystems*, *Java*, allows such functionality. This paper will not focus on Network Management, but will assume that the reader has at least a basic knowledge. The paper will take time to emphasize that Network Management is more than just fault management. There are five areas of network management as defined by the International Organization for Standardization Network Forum: fault management, configuration management, performance management, security management, and accounting management

1.1 Thesis

This primary objective of this thesis was to develop a hypertext system to process and visualize network management data. The process collecting information (e.g., SNMP) executes on different client machines. The client machines are connected to a

server by hyperlinks (e.g., WWW). Using Java, the *Distributed Reporting Management System* (DRMS) application executes code on the clients and returns meaningful results from raw data that is processed.

1.2 Organization

This thesis is organized into the following chapters:

- Chapter II: Overview of Simple Network Management Protocol (SNMP)
- Chapter III: Overview of Java
- Chapter IV: Design and Implementation Issues of Distributed Reporting Management System (DRMS)
- Chapter V: Summary and Future Work

CHAPTER II

SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP)

2.1 Background

SNMP was originally developed as a practical, simple method for managing bridges and routers in networks using TCP/IP [Stallings 93]. Use of SNMP has far exceeded its designers' expectations. In the US, it has become the most widely implemented, non-proprietary protocol, for transporting management data between networked devices and the systems that monitor and control these devices.

SNMP's primary advantages are that it is an industry standard, it can flexibly support vendor-specific extensions, and it is relatively inexpensive to implement. When it arrived in 1988, there was much demand for a way to manage devices in a multi-vendor, multi-platform environment. The International Standards Organization (ISO) was attempting to finalize an Open Systems Interconnection (OSI)-based management standard during this time, but the world had waited long enough. SNMP went from concept to commercial production in a matter of months. Although SNMP has several limitations, it provides enough functionality for the market demand and OSI-based management has declined dramatically.

The primary goal in the development of SNMP was simplicity. One example of this simplicity, is that management data is collected by polling. Polling is a process where the management station periodically sends out requests across the network to determine the status of a device.

2.2 Communications

Managers and agents communicate by exchanging messages that are contained in a User Datagram Protocol (UDP) datagram. Messages must be constructed according to the basic encoding rules of the Abstract Syntax Notation-1 (ASN.1) grammar. SNMP messages consist of a version identifier, an SNMP community name, and a protocol data unit (PDU).

2.3 Commands

SNMP commands are based upon two simple operations: get and set. The get commands are GetRequest, GetNextRequest, and GetResponse. They allow a management station to inspect the Management Information Base (MIB) variables. The SetRequest command allows a management station to change MIB variables. SNMP agents many times support read-only MIB variables which can not be set. This is often used because of the lack of security features available. SNMP also has a trap command. A trap is an unsolicited message that is sent by an SNMP agent to a management station to alert about a specific network event.

2.4 MIBs

SNMP's MIB is flexible enough to support a variety of requirements. SNMP provides a standard, bare-boned implementation. However, it also allows vendors to have

private MIBs which enable them to add the proprietary features that give them the edge in the market place. MIB is in its second version, so is called MIB-II. It is made up of 10 groups as follows:

System Group - provide the name of the managed device (agent), its location, and the person responsible for the device

Interfaces Group - information about each interface that is attached to a subnetwork

Address Translation Group - A combination of several smaller tables that convert network addresses (such as IP addresses) to physical addresses (specific to a subnetwork)

Internet Protocol (IP) Group - information about the implementation and execution of IP

Internet Control Message Protocol (ICMP) Group - information about the implementation and execution of ICMP

Transmission Control Protocol (TCP) Group - information about the implementation and execution of TCP

User Datagram Protocol (UDP) Group - information about the implementation and execution of UDP

Electronic Gateway Protocol (EGP) Group - information about the implementation and execution of EGP

Transmission Group - information about the transmission schemes and access protocols

SNMP Group - information about the implementation and execution of SNMP

An agent does not have to support all of the groups to be SNMP compliant. However, if the agent supports any given group, it must support all variables in that group. This helps the SNMP manager know what it can expect when it seeks to query for values.

2.5 Current Status

It is becoming rare for vendors to introduce communications equipment and software that is not SNMP manageable. Hundreds of devices now support SNMP agents;

these include: bridges, routers, wiring hubs, terminal servers, repeaters, network interface cards, file servers, print servers, FDDI components, most PCS, workstations, modems, multiplexers, uninterruptible power supplies (UPSs), and most LAN and internetwork traffic monitors [Simple 95]. Additionally, if the device does not currently support SNMP, there may be a proxy available that will translate the proprietary communication to SNMP and will forward the information on in SNMP format. As a result, SNMP is available for almost all devices.

SNMP and MIB are evolving technologies with new releases continuing to be produced. Because the topic of this paper is to use SNMP as a tool to solve a problem, the paper will not focus any further on enhancements or needs associated with SNMP or MIB.

CHAPTER III

JAVA

3.1 Background

Java is one of the latest programming languages to hit the market and it comes from *Sun Microsystems*. According to Gosling and McGilton, "the Java Programming Language environment provides a portable, interpreted, high-performance, simple, object-oriented programming language and supporting run-time environment" [Gosling and McGilton 95]. The primary reason Java was chosen to be the language used in this thesis was the ability of Java to execute code across a network on a client machine regardless of the client platform.

Java is simply a name chosen to represent liveliness, animation, speed, and interactivity. Unlike many computer terms, the name is not an acronym, but rather a reminder of that hot, aromatic beverage that many programmers enjoy drinking [Java FAQ 95].

3.2 Main Features of the Language

One of Java's development goals was to take the best of C, Objective C, C++, Eiffel, Ada, and "related languages" in order to form a more complete, robust programming tool [Lemay and Perkins 96]. This also minimizes the learning curve that programmers will have to endure.

3.2.1 Primitive Data Types

There are three groups of primitive data types: numeric types, Boolean types, and arrays. Integer numeric data types are as follows:

8-bit	*byte*
16-bit	*short*
32-bit	*int*
64-bit	*long*

Figure 1 -- Java Integer Number Data Types

There is no *unsigned* type specifier for integer data types in Java. The real numeric types are:

32-bit	*float*
64-bit	*double*

Figure 2 -- Java Real Numeric Data Types

According to Gosling and McGilton, Java's *char* data type defines a sixteen-bit unicode character. Unicode characters are unsigned 16-bit values that define character codes in the range through 65,535 [Gosling and McGilton 95].

A Java *Boolean* variable assumes the value *true* or *false*. A Java *Boolean* is a distinct data type; unlike common C practice, a Java *Boolean* type can not be converted to any numeric type.

In contrast to C and C++, Java language *arrays* are first-class language objects. An array in Java is a real object with a run-time representation. It is possible to declare and allocate arrays of any type, and it is possible to locate arrays of arrays to obtain multi-dimensional arrays. To get the length of an array, use the *length*() accessor method on the array object whose length is desired to be known: *myPoints.length*() returns the number of elements in *myPoints* [Gosling and McGilton 95].

3.2.2 Strings

Strings are Java language objects, not pseudo-arrays of characters as in C. There are actually two kinds of string objects: the *String* class is for read-only (immutable) objects. The *StringBuffer* class is for string objects that need to be modified (mutable string objects).

3.2.3 Arithmetic and Relation Operators

All the familiar C and C++ operators apply. Because Java lacks *unsigned* data types, the >>> operator has been added to the language to indicate an unsigned (logical) right shift. Java also uses the + operator for string concatenation.

3.2.4 Multi-Level Break

Java does not have an explicit *goto* statement. To *break* or *continue* multiple-nested loop or switch constructs, place labels on loop and *switch* constructs, and then *break* out of or *continue* to the block named by the label.

3.2.5 Memory Management and Garbage Collection

After being plagued in C++ by memory management problems, Java has come up with a solution to the problem -- it provides its own memory manager. Former memory techniques such as C-style pointers, pointer arithmetic, *malloc*, and *free* do not exist.

Instead, automatic garbage collection is now an integral part of Java and its run-time system. While Java has a *new* operator to allocate memory for objects, there is no explicit *free* function. Once an object has been allocated, the run-time system keeps track of the object's status and automatically reclaims memory when the object is no longer in use, freeing memory for future use.

3.2.6 Multi-Threading

Java provides its own multi-threading object that allows running multiple programs easily. A common use would include playing sound or running animation while scrolling through a page.

According to Gosling and McGilton, the major problem with explicitly programmed thread support is that a person can never be quite sure they have acquired the locks that they need and released them again at the right time [Gosling and McGilton 95]. If a function returns from a method prematurely or if an exception is raised, the lock has not been released; deadlock is usually the result.

The multi-threading allows Java-enabled web browsers to run specially-written helper programs called "applets". This allows the web page to automatically create any extra feature that the browser will require.

3.3 Architecture

According to Gosling and McGilton, the Java compiler doesn't generate "machine code" in the sense of the native hardware instructions--rather, it generates bytecodes: a high-level, machine-independent code for a hypothetical machine that is implemented by the Java interpreter and run-time system [Gosling and McGilton 95].

Bytecodes are an important concept for a couple of key reasons: portability and reliability. Java is so portable, that the source code does not have to be copied over to a different platform and re-compiled. Simply copy the executable over to any platform that has Java installed, and the bytecodes will be translated according to the platform [Gosling and McGilton 95].

The Java compiler promotes reliability by having string compile-time and run-time checking. This guards against version mismatch problems and eliminates the possibility of overwriting memory and corrupting data [Gosling and McGilton 95].

Gosling and McGilton have stated some other advantages of the Java compiler. The advantages are that the compiler is interpreted, it is dynamic, and it defers memory layout until run time. The interpreted environment enables fast prototyping without waiting for the traditional compile and link cycle. Second, the environment is dynamically extensible, whereby classes are loaded on the fly, as required. Finally, because Java defers memory layout until run time, constant recompilation problems are avoided [Gosling and McGilton 95].

3.4 Security

Security is of utmost importance in a system that can run code on another person's machine; especially with the fear of viruses and Trojan horses running around. Java provides security in four ways: providing its own memory allocation and layout, running a bytecode verifier, running a bytecode loader, and having a networking package to control access.

3.4.1 Memory Allocation and Layout

Gosling and McGilton state several ways that Java uses its memory manager as a first line of defense in security. First, the layout of memory is deferred to run time, and

will potentially differ depending on the characteristics of the hardware and software platforms on which the Java system executes. Second, the Java compiled code references memory via symbolic "handles" are resolved to real memory addresses at run time by the Java interpreter [Gosling and McGilton 95]. This means that there are no pointers in Java, which developers could use to directly manipulate memory. Although Java is based on C++, being able to directly manipulate memory is a dangerous feature which is not included [Gosling and McGilton 95].

3.4.2 Byte Code Verifier

Double checking the use of Java's own bytecodes can provide information to detect security violations. According to Gosling and McGilton, the Java bytecode loader and verifier make no assumptions about the primary source of the bytecode stream--the code may have come from the local system, or it may have traveled halfway around the planet [Gosling and McGilton 95]. The bytecode verifier acts as sort of a gatekeeper: it ensures that code passed to the Java interpreter is in a suitable state to be executed and can be run without fear of breaking the Java interpreter.

The bytecode verifier specifically checks for the following [Gosling and McGilton 95] :

- There are no operand stack overflows or underflows
- The types of parameters of all bytecode instructions are known to always be correct
- Object field accesses are known to be legal--private, public, or protected

3.4.3 Byte Code Loader

The third level of protection is in the class loader and is a feature of the run-time environment. The class loader ensures that built-in functions are not replaced by code downloaded from the network. An example of a built-in function would be those used for file access and other system-specific functions.

More specifically, the java.lang.ClassLoader class defines the way that Java classes are loaded over the network. Applet viewers and web browsers create subclasses of this class. The class implements security policies and defines how class files are loaded via various protocols. One important function of the class loader is to ensure that loaded classes reside in a separate namespace than classes loaded from the local system. This prevents naming conflicts, and also prevents a malicious applet from replacing standard Java classes with its own versions. The class loader is an important part of security.

3.4.4 Networking Java

The networking package from Java is the front-line of defense for security at the network interface level. It controls access to the local disk and can optionally limit applications from accessing machines over the network. The package can be set-up in the following ways:

- Disallow all network accesses
- Allow network accesses to only the hosts from which the code was imported
- Allow network accesses only outside the firewall if the code came from outside
- Allow all network accesses

3.4.5 Types of Java

Java Applets are small applications that are run from a Java-enabled browser and have quite a few security restrictions. By running inside a browser, applets have access to the same capabilities that a browser has: sophisticated graphics, drawings, and image processing packages; user interface elements; networking; and event handling. Java applications can also take advantage of these features, but they do not require them.

According to David Flanagan, different web browsers and applet viewers may place slightly different restriction on applets [Flanagan 96]. Some may even allow the user to relax some of these restrictions, but, in general, applets are NOT allowed to:

- Read files on the local system
- Write files to the local system
- Delete files on the local system
- Rename files on the local system
- Create a directory on the local system
- List directory contents
- Check for the existence of a file
- Obtain the type, size, or modification time of a file
- Create a network connection to any computer other than the one from which the Applet was itself loaded.
- Listen for or accept network connections on any port of the local system
- Create a top-level window without a visible warning indicator that the window is "untrusted". This prevents applets from spoofing other trusted programs into which users may type sensitive data.
- Obtain the user's username or home directory name
- Define any system properties
- Invoke any program on the local system

- Access or load classes in any package other than the standard eight that come with the Java API.
- Define classes that are part of the package on the local system.

And, the list goes on. According to Lemay and Perkins [Lemay and Perkins 96], "these restrictions can prevent most of the more obvious ways of trying to cause damage to a client's system, but it's impossible to be absolutely sure that a clever programmer cannot somehow work around these restrictions." Some of the ways to "work around the restrictions" will be explained in the installation details of the *DRMS* application in Section 4.1.

Java applications are stand-alone Java programs that can be run by using just the Java interpreter, for example, from a command line. At first glance, the distinguishing factor between a Java application and a Java applet is that a Java application has a main() function.

JavaScript is not Java. However, it is based on the Java language. JavaScript was written by Netscape and is a programmable API that allows cross-platform scripting of events, objects, and actions [Lemay and Perkins 96]. It allows the page designer to access events such as startups, exits, and user mouse clicks. JavaScript extends the programmatic capabilities of Netscape Navigator to a wide range of authors and is easy enough for anyone who can compose HTML. One could use JavaScript to glue HTML, in-line plug-ins, and Java applets to each other.

3.4.6 Denial of Service Attacks

According to David Flanagan [Flanagan 96], the one "security hole" that remains when running an untrusted applet is that the applet can perform a "denial of service attack" on your computer. For example, it could frivolously allocate lots of memory, run

many threads, or download lots of data. This sort of attack consumes system resources and can slow your computer (or your network connection) considerably. While this sort of attack by an applet is inconvenient, fortunately it cannot do any real damage (except to your patience and your deadlines.)

3.4.7 Applet Certification

One hope for the future is that being able to verify the source of an applet will reduce attacks. David Flanagan states that a possibility being explored for future releases of Java is the ability for applets to carry an attached digital signature and cryptologic checksum [Flanagan 96]. Using public key encryption technology, this would enable a Web browser or applet viewer to verify that an applet is from the source it claims to be from, and that it has not been modified in transmission. With these guarantees, it is possible to load a trusted applet (one that can run without severe security restrictions) over an untrusted network as long as the source of the applet is trusted (i.e., whatever individual encryption or certification agency has attached their digital signature to the applet.)

3.5 Java Executables

Java Development Kit (JDK) is available free of charge. The caveat is that when a Java application is sold, part of the proceeds must go back to Sun Microsystems.

A quick list of the most used executables that come with JDK are as follows [Flanagan 96]:

- appletviewer -- The Java Applet Viewer
- java -- The Java Interpreter
- javac -- The Java Compiler
- javadoc -- The Java Documentation Generator

- jdb -- The Java Debugger

Each of the applications have various parameters with which they can be executed. For more information refer to the Java *Sun Microsystems* home page (http://www.sun.com/JDK-1.0/) on the Internet.

CHAPTER IV

DESIGN AND IMPLEMENTATION ISSUES

4.1 Implementation of SNMP and JAVA

The implementation of the DRMS application will require five parts. First an SNMP agent will need to be running on the client machine. The second part will be either the Java run-time or Java-enabled Netscape. The third part is the SNMP Java classes, and the fourth part is the DRMS Java applet classes. The final part, a SNMP applet Server (SAS), will be only be needed if the second and third parts are located on the web server.

The SNMP agent that DRMS uses supports SNMP version 1. The MIB that DRMS uses is defined in RFC 1213. Because of the use of the SNMP Java classes, it would be easy to expand DRMS to use other MIBs, however, for the purpose of this demonstration, the use of one MIB is sufficient. In the testing that follows, the Windows 95 SNMP agent is used; it can be located on the Windows 95 CD ROM or from the Microsoft home page (http://www.microsoft.com) on the Internet.

Since DRMS is a relatively small program, in this case less than 1700 lines of code, it is written as a Java Applet. As a result, to run DRMS a Java-enabled browser is required. Although DRMS will run under Netscape, it is significantly slower than running it under the Java AppletViewer. Netscape is available without charge to students and can be downloaded from the Netscape Internet home page (http://www.netscape.com). As mentioned earlier, Java does not require a payment unless a Java program is sold. Java can be downloaded from the Java Internet home page (http://java.sun.com).

Another advantage of using the Java AppletViewer is that it is easy to size the screen of the application that is running; this involves re-sizing the window just as one can do for most windows applications. When running Netscape, Java applets do not dynamically size to the screen. Consequently, depending on the size of the application, it may be too small on the screen or may run off of the screen in a manner that causes the user to have to scroll to see the application. An additional concern with running Netscape is that there seems to be some inconsistency across platforms in Netscape's support of socket access for local classes.

The SNMP classes are required to interface with the SNMP agent. The classes used by DRMS were written by Advent Network Management, Inc. and can be obtained at the Advent home page (http://www.adventnet.com).

The DRMS applet is the focus of this paper and can be located via anonymous ftp at a.cs.okstate.edu in (/d/usr/gilman/drms). DRMS is comprised of several main classes: the gui (sampleSNMP.class), the interface between gui and graphing (snmpFrame.class), the graphing routine (simpleGraph.class), and a an interface between graphing and SNMP class (snmpGraph.class). The Advent Network Management application did not exist when the development of DRMS began; however it was developed concurrently. The DRMS SNMP Class (snmpGraph.class) provides only a subset of the functionality that the Advent Network Management application would provide should the DRMS application be expanded at a later date.

If the Java-enabled browser and the SNMP classes are not local, then SAS must be installed to overcome normal applet restrictions. SAS is installed on the web server and allows applets to send and receive SNMP packets to and from any managed devices accessible for the applet host. Additionally, SAS allows saving of files in a SASusers sub-directory of the applet directory on the web server. Although this option is available, it is not recommended because it is so slow. SAS can be obtained from the Advent Network Management Home Page (http://www. adventnet.com).

There are three scenarios under which DRMS was written to be implemented. One scenario is that all of the classes are local on a client machine. The advantage of this is that the applet is very fast, there is minimal network traffic and there are no concerns that user has the appropriate rights to files. The disadvantage is that all of the software has to be installed on every machine and if the applet needs to be updated, it has to be updated on every machine. From the time that the application is started to the time that the initial screen is painted, takes an average of 15 seconds. Additionally, if the client is checking its own MIB, there are no packets sent across the network.

The second scenario is that all of the classes are located on a web server. This provides easy installs and upgrades, but the applications will run very slow with significant network traffic. Additionally, there will be concerns about making sure that users have the correct rights to files. Furthermore, users will only be able to save their reporting data on the web server. Finally, from personal experience, the more code that is located on the network, the more difficulty an analyst may have if trying to troubleshoot a network problem.

The third scenario is somewhat of a compromise from the first two scenarios; the Java and SNMP classes are installed locally and the applet is on the web server. The advantages are that this is faster than having all of the classes on the server, and the applet is probably going to have more updates than the other classes; this still provides an easy way to update the applet. The disadvantage is that the browser and SNMP classes have to be installed on the local client. Please note that when this paper refers to the "client machine", it is simply referring to the machine on which the applet is executing; there are some cases where "the client machine" may actually be a server.

Just to load the DRMS from a web server using the third scenario, 149 packets are passed across the network. Additionally, DRMS takes an average of 55 seconds to paint the screen. This was tested a dozen times using token-ring and ethernet during off-prime hours on a private network. If PPP was being used or if this test was conducted across

the Internet, it would probably take longer. The time to load the program and the extra traffic on the network are big concerns for Java considering that this is a small application. However, depending on the requirements for a Java program, these disadvantages may be out-weighed by the advantages of Java, such as the variety of platforms on which the program can run.

To give an idea of the size of the items mentioned above, please find the names of the programs and their associated sizes listed in the table below:

Full Java Development Kit for Windows 95	11,173,888
Netscape 2.01 for Windows 95	7,536,640
SNMP Classes from Advent	409,600
MIB	105,809
DRMS	30,490
SAS	11,789

Figure 3 – Software Required for DRMS

Once the previous software has been obtained, please refer to Appendix C for a DRMS User's guide that includes how to install the application.

4.2 User Interface

The User Interface, as seen below, is very simple. The text area is used for displaying messages and status. After pressing the help button, the help screen appears. The help screen serves two purposes; initially it provides information to the user about the

available functions. However, if the screen is left open, it provides DEBUG information on what is actually happening behind the scenes.

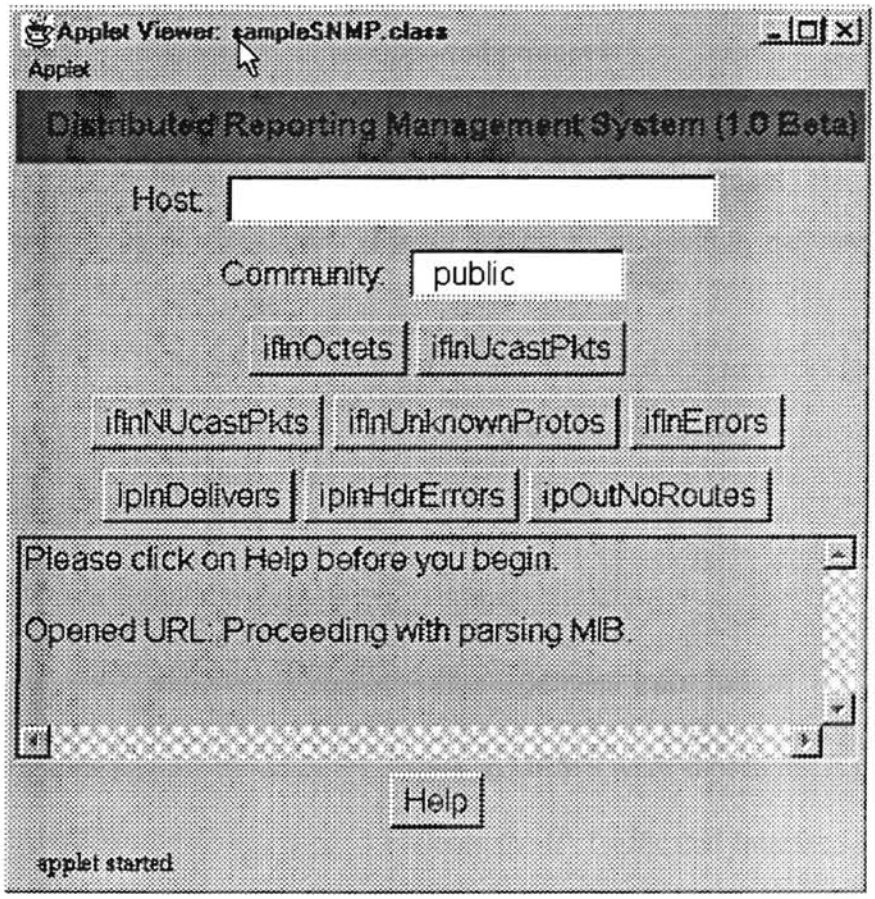

Figure 4 – DRMS Main Screen

To view information from a MIB, one must have an SNMP agent running. Enter an IP address and perhaps a community and choose one of eight graphical displays. The eight choices are listed below:

ifInOctets - Total number of Octets (bytes) received on the interface, including framing characters

ifInUcastPkts - Number of subnetwork-unicast packets delivered to a higher-level protocol

ifInNUcastPkts - Number of nonunicast packets delivered to a higher-layer protocol

ifInUnknownProtos - Number of inbound packets that were discarded because of an unknown or unsupported protocol

ifInErrors - Number of inbound packets that contained errors preventing them from being deliverable to a higher-level protocol

ipInDelivers - Total number of input datagrams successfully delivered to IP user protocols

ipInHdrErrors - Number of input datagrams discarded due to errors in IP header.

ipOutNoRoutes - Number of IP datagrams discarded because no route could be found

4.3 Display of Network Management Information

The graphs that are available for display in DRMS were chosen because these were the values that William Stallings used to evaluate the IP Traffic and Interface-Level Traffic in his book in Tables 5.10 and 5.11 [Stallings 93]. Not all of the graphs are meaningful in that if there are not many errors, then the graph is just a line along the bottom of the graph. However, it could help diagnose a healthy network and if a person was trouble-shooting errors, the DEBUG section in the help screen might offer valuable information to be able to correct the problem.

The two most active graphs are *ipInOctets* and *ipInDelivers*. These two sample graphs are shown in Figures 5 and 6. These graphs were taken at the same time, running DRMS off a HP-UX Web Server, while the other classes were running locally on a

Windows95 system. In this example, the network is Token Ring and the only applications running besides DRMS are Netscape and ftp. The Netscape session is viewing home pages from the Internet.

In the upper right-hand corner of the graph in Figures 5 and 6, the host in which DRMS was connected is listed. Also listed is the OID (object identifier) that was queried from the MIB.

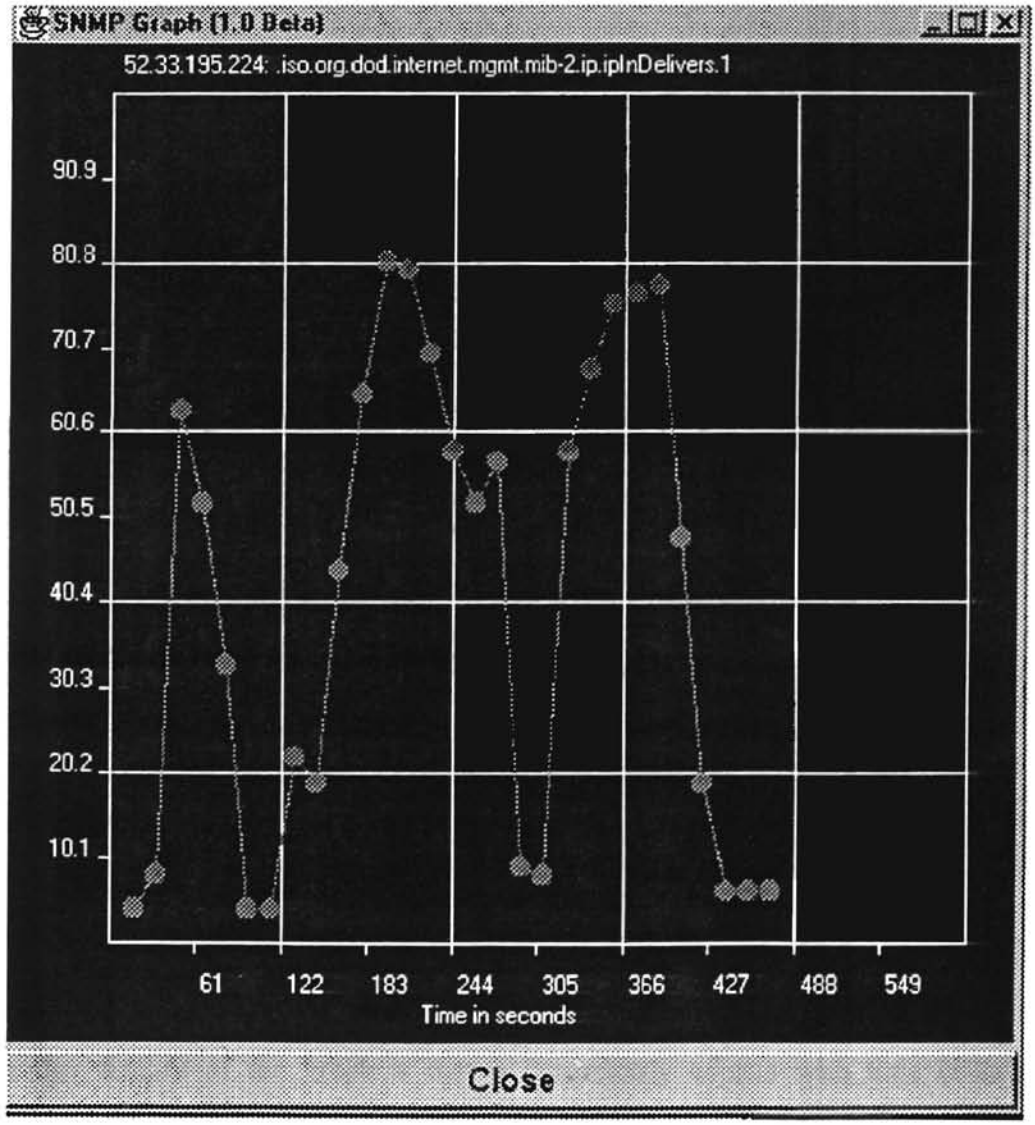

Figure 5 – Graph of inIPDelivers

As indicated by the OID in Figure 5, the numbers along the left-side of the screen indicate the number of IP packets that were received. The numbers along the bottom of the screen list the time in seconds. Based on this information, it is easy to see that the graph is depicting the throughput.

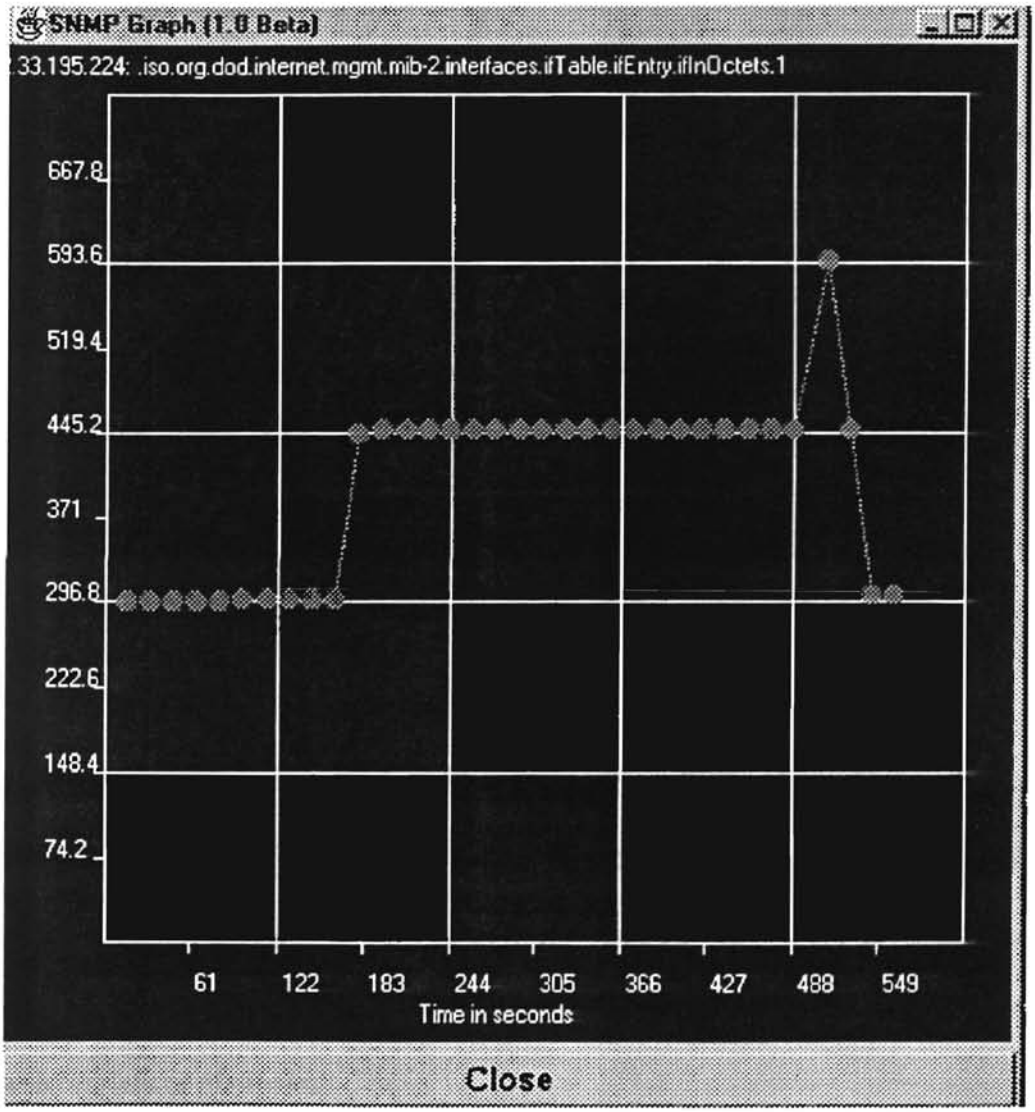

Figure 6 -- Graph of ifInOctets

Another feature of the graph is that it dynamically adjusts with time (x-axis). When the graph first appears, it shows the packets arriving in 30 second intervals for a

maximum of 300 seconds. In the graphs in figures 5 and 6, the time surpassed 300 seconds, therefore the graph adjusted to meet the data requirements. The y axis is dynamic as well.

As indicated by the OID in figure 6, the numbers along the left-side of the screen indicate the number of IP octets (bytes) that were received.

4.4 Observations of the Tool

DRMS has been implemented and tested on Token-Ring, Ethernet, and serial link (PPP) over a private intranet. Packet traffic was monitored with HP Openview to ensure accuracy of the DRMS application. Additionally, a dozen colleagues were consulted and encouraged to test the package on individual workstations. Based on these observations, the help screen in the application was improved the following list of advantages and disadvantages where compiled.

4.4.1 Advantages of DRMS

DRMS has a simple, graphical interface and visualization tools which would indicate a small learning curve when beginning to use the tool. Additionally the help screen and the debugging feature, help the user to accomplish their work more easily.

Being able to perform distributed network management is a big advantage to those that are trying to administer a system remotely. Since DRMS is written in Java, it can run on a wide variety of platforms.

4.4.2 Concerns about DRMS

Although SNMP is a standard among network manageable interfaces, it lacks security features that will prevent this version of DRMS from being more functional. Additionally, even though there are features of Java that appear to give it the edge for an exciting future over other operating systems, the DRMS application brings several Java issues to view:

- Java is slow
- When run across the network, Java creates much network traffic
- Java is not available for Windows 3.1
- This type of application might be more useful for a smaller network because often times big corporations have total systems that are dedicated to this type of monitoring.

CHAPTER V

SUMMARY AND FUTURE WORK

The primary objective of this thesis was to develop a network management tool: a hypertext system to process and visualize management data across a network. The application collecting information (e.g., SNMP) executed on different client machines. The client machines were connected to a server by hyperlinks (e.g., WWW). Using Java, the *Distributed Reporting Management System* (DRMS) application executes code on the clients and returns meaningful results from raw data that has been processed.

In Chapter I, there was an introduction. Chapters II and III gave the background for SNMP and Java, respectively. Chapter IV provided the specifics of the DRMS application.

The conclusion is that the only time that one would want to run the Java classes from a remote site is when the value of executing remotely outweighs the bandwidth and time required to run the program. In general, if the processing resources required of the program outweigh the network traffic required to execute the program, then this would be a good situation to consider. In the DRMS example, a graph was chosen to test the processing of the raw data. In real life, this is a poor example in that the graph itself would probably never be sent across the network back to the client, probably only the raw data would be sent.

If the entire program is chosen to be done locally, then there is a savings on both processing speed and network traffic, however a person will loose the advantage of being able to update the executables easily in one central location.

This application served as a good exercise to learn Java & SNMP and to investigate using them together in running network management applications across the network. The DRMS code and this paper will be provided to Advent Network to allow them to incorporate these findings into their commercial products.

There are many possible enhancements that could be added to future releases of DRMS:

- Currently DRMS only gathers data from a RFC 1213-MIB, allow other MIBs.
- Allow printing from within the application.
- Allow saving information for historical reference.
- Allow combining multiple reports to produce an executive summary.
- Use other SNMP functions to allow further network management (Set & Trap).
- Allow SNMP functions on a wider range of MIB variables.

This DRMS application is a benefit to computer science because it shows that remote network management will be available in the near future. The concerns around Java will be corrected with new releases of the software and the concept of DRMS will be readily implementable. DRMS is just the beginning of many new releases of hypertext systems to process and visualize network management data.

References

[Advent 96] Advent Network Management, Inc., WWW(http://www.adventnet.com/), 1996.

[Buerger 95] David Buerger, "The feel-good IETF process is due for a change", *Network World,* Vol. 12, No. 48, p. 74, November 1995.

[Coopee 95] Todd Coopee, "Covering the Network Management Spectrum", *Network World,* Vol. 12, No. 22, pp. 57-58, May 1995.

[Comer 95] Douglas Comer, *Internetworking with TCP/IP*, Vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1995.

[Comer 91] Douglas Comer, *Internetworking with TCP/IP*, Vol. 2, Prentice-Hall, Englewood Cliffs, NJ, 1991.

[Cortese 95] Amy Cortese, "The software revolution", *Business Week*, No. 3453, pp. 78-85, December 1995.

[Dodge 95] John Dodge, "Growing pains in the Internet World", *PC Week*, Vol. 12, No. 483, p. 3, October 1995.

[Farber 95] Dan Farber, "Microsoft gives nod to Sun's Java", *PC Week,* Vol. 12, No. 49, p. 138, December 1995.

[Flanagan 96] David Flanagan, "Java in a Nutshell", O'Reilly & Associates, Inc., Sebastopol, CA, 1996.

[Freeman 96] Paul Freeman, "Paul Freeman Associates, Inc.: SNMP Agent Tools for Windows NT and Unix", WWW(http://world.std.com/~pfa), 1996.

[Gonzalez 95] Sean Gonzalez, "Java & HotJava: waking up the Web", *PC Magazine,* Vol. 14, No. 18, pp. 265-266, October 1995.

[Gosling and McGilton 95] James Gosling and Henry McGilton, "The Java (tm) Language Environment: A White Paper",WWW(http://java.sun.com/whitePaper), 1995.

[Hunt 95] Craig Hunt, *Networking Personal Computers with TCP/IP*, O'Reilly & Associates, Inc., Sebastopol, CA, 1995.

[Java 95] "Java: Metaphor for distributed computing", *CommunicationsWeek*, Vol. 582, p. 32, October 1995.

[Java FAQ 95], "Java and HotJava FAQ",WWW(http://www.javasoft.com/ faq2. html#A5), 1995.

[Java: Programming 96], "Java: Programming for the Internet", WWW(http://java.sun. com/), 1996.

[Leinwald 96] Allan Leinwald, *Network Management*, Addison-Wesley, Reading, MA, 1996.

[Lemay and Perkins 96] Laura Lemay and Charles Perkins, *teach yourself JAVA in 21 days*, Sams.net, Indianapolis, IN, 1996.

[Katt 95] Spenser Katt, "IBM and Java go together like Norway and Easter", *PC Week*, Vol. 12, No. 49, p. 138, December 11, 1995.

[McConnell 95] John McConnell, "Spectrum Test Drive revs up SNMP platform competition", *Network World*, Vol. 12, No. 27, p. 29, July 1995.

[Mier 95a] Edwin Mier, "Waiting for SNMPv2? Make SNMP secure in the meantime", *CommunicationsWeek*, Vol. 562, p. 25, June 1995.

[Mier 95b] Edwin Mier, "All you ever want to know about SNMP in the year to come", *CommunicationsWeek*, Vol. 588, p. 29, December 1995.

[Mier, Mier, and Yocom 95] Edwin Mier, David Mier, and Betsy Yocom, "Simply marvelous", *CommunicationsWeek*, Vol. 543, pp. 55-59, February 1995.

[Miller 95] Michael Miller, "Casting a net beyond HTML", *PC Magazine*, Vol. 14, No. 20, pp. 75-77, November 1995.

[NMS 96] "NMS: Product Reviews",WWW(http://smurfland.cit.buffalo.edu/NetMan/ ProductReviews.html), 1996.

[Norr 95] Henry Norr, "Sun prompting industry to wake up, smell the Java", *MacWeek*, Vol. 9, No. 42, p. 28, October 1995.

[O'Brien 95] Jim O'Brien, "Java animates the Web with interactive content", *Computer Shopper*, Vol. 15, No. 12, pp. 655-656, December 1995.

[OSCOM 96] OSCOM: MIBmanage, WWW(http://www.oscom.com.au/ mibmanage.html), 1996.

[OutBack 96] OutBack Resource Group, Inc., WWW(http://www.outbackinc.com/), 1995.

[Pang and Yamada 95] Albert Pang and Ken Yamada, "Leverage Java, channel: Sun's new desktop chief outlines software, partner plans", *Computer Reseller News*, Vol. 658, p. 497, November 1995.

[Pedersen 95] Elinor Pedersen, "Homogeneous ATM is one legacy away", *Midrange Systems*, Vol 8, No. 17, p. 30, September 1995.

[Shevenell 95] Michael Shevenell, "Needed: A foundation for apps management", *Network World*, Vol. 12, No. 11, p. 51, March 1995.

[Simple 95] "Simple Network Management Protocol (SNMP)", Faulkner FACCTS: The Ultimate Resource, Pennsauken, NJ, November 1995.

[Singh 95] Jai Singh, "With enough Java it seems pigs could fly", *InfoWorld*, Vol. 17, No. 50, p. 3, December 1995.

[Smith 95] Tom Smith, "Java, GTSI, Delrina brew up hot new home pages", *Computer Reseller News*, Vol. 659, p. 12, November 1995.

[Stallings 93] William Stallings, *SNMP, SNMPv2, and CMIP: The practical guide to network management standards*, Addison-Wesley, Reading, MA, 1993.

[Staten 95] James Staten, "Java brewing for Mac; development kit builds Java applets", *MacWeek*, Vol. 9, No. 42, p. 4, October 1995.

[Sullivan 95a] Eamonn Sullivan, "Sun's Java technology perks up WWW; Java language and HotJava browser provide extensibility to the World-Wide-Web", *PC Week*, Vol. 12, No. 22, p. 14, June 1995.

[Sullivan 95b] Eamonn Sullivan, "Java reality short of hype: SDK in need of better programming environment", *PC Week*, Vol. 12, No. 45, p. 220, November 1995.

[Sullivan 95c] Eamonn Sullivan, "Beyond Java: distributed objects on Web", *PC Week*, Vol. 12, No. 50, p. 48, December 1995.

[Townsend 95] Robert Townsend, *SNMP Application Developer's Guide*, Van Nostrand Reinhold, Milford, CT, 1995.

[Voss 95] Greg Voss, "Holes in the Bucket", *Windows Tech Journal*, Vol. 4, No. 11, pp. 61-63, November 1995.

[Weston 95] Rusty Weston, "Sun's software chief sheds light on Unix, Java and Windows NT", *PC Week*, Vol. 12, No. 47, pp. 22-23, November 1995.

[Wong 95] William Wong, "A window with a view", *LAN Magazine*, Vol. 10, No. 1, pp. 126-132, January 1995.

Appendix A: Glossary

API (Application Programming Interface) - A programmatic interface to a set of software libraries.

Applet - a dynamic and interactive program that can run inside a Web page displayed by a Java-capable browser such as HotJava or Netscape 2.0.

ASN.1 (Abstract Syntax Notion One) - The OSI language for describing abstract syntax.

CCITT (International Telephone and Telegraph Consultative Committee) - An international organization that defines standards and recommendations for the connection of telephone equipment. Now known as the ITU-T.

DRMS (Distributed Reporting Management System) - the topic of this thesis; uses Java and SNMP to perform networking reporting over a distributed environment.

EGP (Exterior Gateway Protocol) - A routing protocol for passing reachability information between autonomous systems; document in RFC 904.

HTML (Hyper-Text Media Language) - The standard language used for creating web pages.

ICMP (Internet Control Message Protocol) - A network layer protocol that carries messages to report errors relevant to IP packet processing; documented in RFC 792.

Internet - A term used to refer to the world's largest internetwork, which connects thousands of networks around the world. It is usually referenced as "the Internet."

IP (Internet Protocol) - A network layer protocol that contains addressing and control information for packets to be routed; documented in RFC 791.

ISO (International Organization for Standardization) An international body that develops, suggests, and names standards for network protocols.

ITU-T (International Telecommunication Union Telecommunication standardization sector). See CCITT.

JDK (Java Development Kit) -- The executables and utilities used to compile and run Java programs.

MIB (Management Information Base) - A database of managed objects accessed by network management protocols.

MIB-I (Management Information Base - I) The first MIB defined for managing TCP/IP-based internets; documented in RFC 1156.

MIB-II (Management Information Base - II) The current standard MIB defined for managing TCP/IP-based internets; documented in RFC 1213.

packet - a logical collection of data.

PDU (Protocol Data Unit) - a sequence of blocks that is used to exchange streams of data between two entities.

performance management - The process of analyzing the characteristics of a data network to monitor and increase its efficiency.

OID (Object identifier) - a series of integers separated by periods that denote the exact traversal of a MIB tree to a labeled node.

OSI (Open Systems Interconnection) - An international network protocol suite developed by ISO and ITU-T (formerly CCITT) for use on multi--vendor equipment.

RFC (Request for Comments) - Documents to communicate ideas for development in the Internet community and that might become Internet standards.

SNMP (Simple Network Management Protocol) - A network management protocol used for managing network devices; documented in RFC 1448.

TCP (Transport Control Protocol) - A transport layer protocol that provides reliable transmission of data on IP networks; documented in RFC 793.

TCP/IP - The two most popular Internet protocols that provide Transport Layer and Network Layer service.

Throughput - The rate of information arriving at, and possibly going through, a point in a network system.

UDP (User Datagram Protocol) - A connectionless transport protocol used on IP networks; documented in RFC 768.

Web Pages - Hypertext documents that are found on the WWW.

WWW (World Wide Web) - The large-scale information service that allows a user to browse information. (a.k.a. the graphical portion of the Internet) WWW offers a hyper media system that can store information as text, graphics, audio, etc..

Appendix B: Trademark Information

Advent Network Management is a registered trademark of Advent Network Management, Inc.

JAVA is a registered trademark of Sun Microsystems, Inc.

Appendix C: DRMS User's Guide

C.1 Implementation and Installation of DRMS

The implementation of the DRMS application requires five parts. First an SNMP agent needs to be running on the client machine. The second part is either the Java run-time or Java-enabled Netscape. The third part is the SNMP Java classes, and the fourth part is the DRMS Java applet classes. The final part, a SNMP applet Server (SAS), will be only be needed if the second and third parts are located on the web server.

The SNMP agent that DRMS uses supports SNMP version 1. The MIB that DRMS uses is defined in RFC 1213. Because of the use of the SNMP Java classes, it would be easy to expand DRMS to use other MIBs, however, for the purpose of this demonstration, the use of one MIB is sufficient. In the testing that follows, the Windows 95 SNMP agent is used; it can be located on the Windows 95 CD ROM or from the Microsoft home page (http://www.microsoft.com) on the Internet.

Since DRMS is a relatively small program, in this case less than 1700 lines of code, it is written as a Java Applet. As a result, to run DRMS a Java-enabled browser is required. Although DRMS will run under Netscape, it is significantly slower than running it under the Java AppletViewer. Netscape is available without charge to students and can be downloaded from the Netscape Internet home page (http://www.netscape.com). As

mentioned earlier, Java does not require a payment unless a Java program is sold. Java can be downloaded from the Java Internet home page (http://java.sun.com).

Another advantage of using the Java AppletViewer is that it is easy to size the screen of the application that is running; this involves re-sizing the window just as one can do for most windows applications. When running Netscape, Java applets do not dynamically size to the screen. Consequently, depending on the size of the application, it may be too small on the screen or may run off of the screen in a manner that causes the user to have to scroll to see the application. An additional concern with running Netscape is that there seems to be some inconsistency across platforms in Netscape's support of socket access for local classes.

The SNMP classes are required to interface with the SNMP agent. The classes used by DRMS were written by Advent Network Management, Inc. and can be obtained at the Advent home page (http://www.adventnet.com).

The DRMS applet is the focus of this paper and can be located via anonymous ftp at a.cs.okstate.edu in (/d/usr/gilman/drms). DRMS is comprised of several main classes: the gui (sampleSNMP.class), the interface between gui and graphing (snmpFrame.class), the graphing routine (simpleGraph.class), and a an interface between graphing and SNMP class (snmpGraph.class). The Advent Network Management application did not exist when the development of DRMS began; however it was developed concurrently. The DRMS SNMP Class (snmpGraph.class) provides only a subset of the functionality that the Advent Network Management application would provide should the DRMS application be expanded at a later date.

If the Java-enabled browser and the SNMP classes are not local, then SAS must be installed to overcome normal applet restrictions. SAS is installed on the web server and allows applets to send and receive SNMP packets to and from any managed devices accessible for the applet host. Additionally, SAS allows saving of files in a SASusers sub-directory of the applet directory on the web server. Although this option is available, it is

not recommended because it is so slow. SAS can be obtained from the Advent Network Management Home Page (http://www. adventnet.com).

There are three scenarios under which DRMS was written to be implemented. One scenario is that all of the classes are local on a client machine. The second scenario is that all of the classes are located on a web server. The third scenario is somewhat of a compromise from the first two scenarios; the Java and SNMP classes are installed locally and the applet is on the web server. For advantages and concerns, please see section 4.1 of this thesis. The fastest execution of DRMS occurs when the entire DRMS is installed locally.

Once the needed applications are installed, the environment variable CLASSPATH needs to be set to point to the browser Java classes and the SNMP applications. In DOS running JAVA, the set statement would look like this:

set classpath=c:\windows\java\lib\classes.zip;c:\snmpjava\prog\classes;.

Where c:\windows\java\lib\classes.zip is the location of the Java classes and c:\snmpjava\prog\classes is the location of the SNMP classes from Advent. In DOS running Netscape, the set statement would look like this:

set classpath=c:\nsnav2\Program\java\classes\MOZ2_01.ZIP;c:\snmpjava\prog\classes;.

Where c:\nsnav2\Program\java\classes\MOZ2_01.ZIP is the location of the Netscape Java classes. Notice that the CLASSPATH ends with a period (.). This allows the current directory to be in the CLASSPATH.

If the DRMS application is installed locally, it is a good idea to add the DRMS application subdirectory to the CLASSPATH too. In this case, the set statement would look like:

set classpath=c:\nsnav2\Program\java\classes\MOZ2_01.ZIP;c:\snmpjava\prog\classes;c:\snmpjava\prog\drms;.

To begin running the applet using the Java AppletViewer, it is recommended that the Java binary subdirectory is in the path. In this case, c:\windows\java\bin would be put in the path. Go to the subdirectory that contains the DRMS applet and classes and type at the prompt> appletviewer drms.html

To begin running the applet using Netscape, start Netscape and do a file open on drms.html that is located in the DRMS subdirectory. In this case, it is http://www.tcom.dupont.com/DRMS/drms.html.

C.2 User Interface

The User Interface, as seen below, is straight forward.

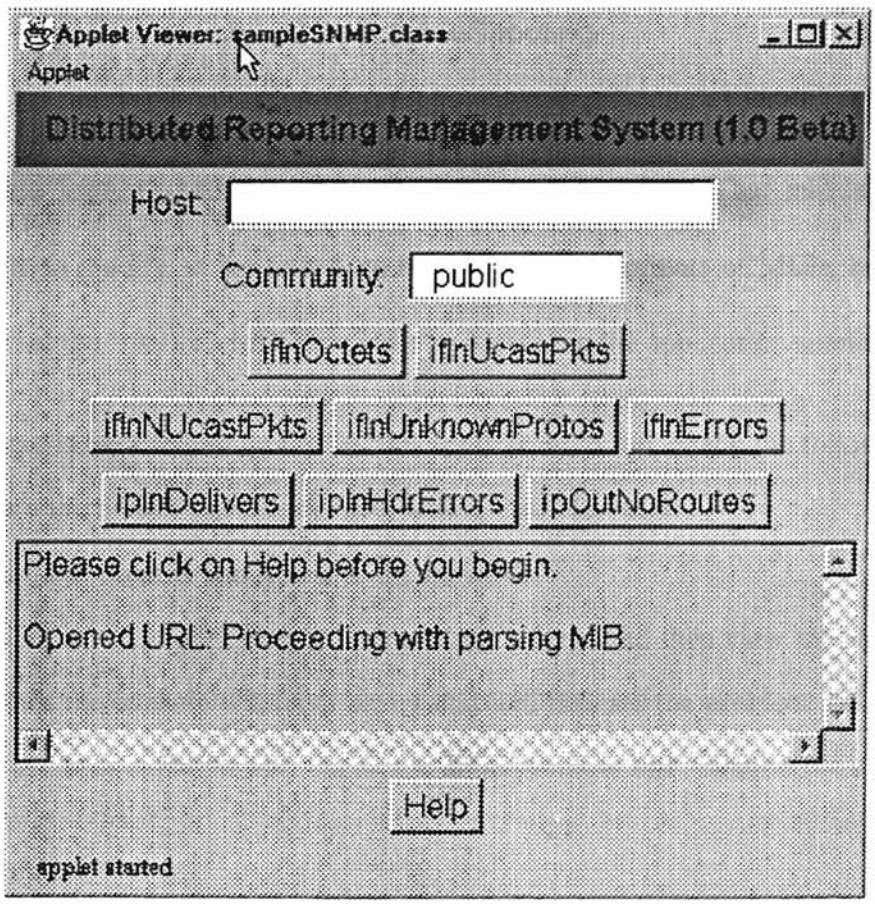

Figure 7 – DRMS Main Screen

To view information from a MIB, one must have an SNMP agent running. Enter an IP address and perhaps a community and choose one of eight graphical displays.

The text area is used for displaying messages and status. After pressing the help button, the help screen appears. The help screen serves two purposes; initially it provides information to the user about the available functions. However, if the screen is left open, it provides DEBUG information on what is actually happening behind the scenes.

To close a graph, click on the close button at the bottom of the graph. To close the DRMS application, click the X button in the upper right-hand corner of the DRMS screen.

C.3 TroubleShooting

The SNMP agent is communicating to its own MIB. The DRMS application reads the ASCII file 'rfc1213-MIB' to know the format of the MIB. That makes the DRMS program easy to modify, in the future, to read in different formats of MIBs, however that also means that the 'rfc1213-MIB' file must be located in the application subdirectory.

If there is trouble getting the DRMS application to run, check the following things:

- Make sure the SNMP agent is running with a rfc1213-MIB
- Make sure the client is connected to the network
- If the client is running Netscape, look under Options, and then view the Java console. Many times the console will have errors that will indicate the problem. If the client is running the Java appletviewer, check the DOS window that initiated the appletviewer process.

- Make sure the CLASSPATH variable is set to both the browser zip file (Java or Netscape) and the SNMP classes directory, and the current directory (.).
- Try starting the browser from the DRMS subdirectory.

Appendix D: DRMS JAVA Code

```
/*****************************************************************************/
/*****************************************************************************/
/* Author:  Christine E. Valliere                          SSN:  585-53-2577  */
/*****************************************************************************/
/*                              Master's Thesis                              */
/*****************************************************************************/
/*     Commitee Members:  Dr. Neilsen, Dr. Mayfield, Dr. Samadzadeh          */
/*****************************************************************************/
/*               DISTRIBUTED REPORTING MANAGEMENT SYSTEM (DRMS)              */
/*****************************************************************************/
/* This is an application that usees SNMP and Java to be able to do          */
/* Network management remotely.  This assumes that an SNMP agent is running */
/* on the machine and requires a Java-enabled browser to run.  DRMS also     */
/* requires the SNMP classes that were written by Advent Network Management */
/* before executing the environment variable CLASSPATH must be set to        */
/* include both the Java-enabled browser zipped up class file and the SNMP   */
/* classes from Advent. (e.g., set CLASSPATH=c:\nsnav2\Program\java\classes */
/*\MOZ2_01.ZIP;c:\snmpjava\prog\classes;. )  For more information, please    */
/* refer to my thesis, Appendix C.                                           */
/*****************************************************************************/
/*****************************************************************************/
import java.awt.*;
import java.util.*;
import java.io.*;
import java.applet.*;
import Snmp.*;
import LabeledTextField;
import ButtonPanel;
import Title;
import UrlDialog;
import snmpFrame;

public class sampleSNMP extends Applet {

/** Some of the widgets in the main applet window. */
  TextArea  resultArea;
  LabeledTextField ltf,host,community;
  Title title;

/** The frame extention class Advent Network created for SNMP operations */
  snmpFrame frame;

/** The applets init method sets up the GUI */
  public void init() {


/** The fonts to be used */
   Font font = new Font("Helvetica",Font.PLAIN,16);
   Font fontb = new Font("Helvetica",Font.BOLD,16);

/** Instantiate the snmpFrame */
```

```
    frame = new snmpFrame(this);

/** The Sample SNMP GUI consists of a top, middle & bottom panels */
    Panel topPanel, middlePanel, bottomPanel;

        // Set up panels
    topPanel = new Panel();
    topPanel.setLayout(new BorderLayout());
    middlePanel = new Panel();
    middlePanel.setLayout(new BorderLayout());
    bottomPanel = new Panel();
    bottomPanel.setLayout(new BorderLayout());

        //The main application will be three rows of buttons
        //row 1
    String b1[]={"ifInOctets","ifInUcastPkts"};
    ButtonPanel bp1 = new ButtonPanel(b1);
    bp1.setFont(font);

        //row 2
    String b2[]={"ifInNUcastPkts","ifInUnknownProtos","ifInErrors"};
    ButtonPanel bp2 = new ButtonPanel(b2);
    bp2.setFont(font);

        //row 3
    String b3[]={"ipInDelivers","ipInHdrErrors","ipOutNoRoutes"};
    ButtonPanel bp3 = new ButtonPanel(b3);
    bp3.setFont(font);

        //put buttons on middle panel
    middlePanel.add("North",bp1);
    middlePanel.add("Center",bp2);
    middlePanel.add("South",bp3);


        // Now define a panel for function keys
    String functionb[] = {"Help"};
    ButtonPanel functionBp = new ButtonPanel(functionb);
    functionBp.setFont(font);

        //put buttons on bottom panel
    bottomPanel.add("South",functionBp);


      // First, the top panel elements with BorderLayout
    host = new LabeledTextField("          Host: ", "", 25);
    topPanel.add("Center", host);
    host.setFont(font,font);

        //put the title on the top panel
    title=new Title(fontb);
    topPanel.add("North",title);

    community = new LabeledTextField("                    Community: ",
"public", 10);
    topPanel.add("South", community);
    community.setFont(font,font);

            // Now the bottom elements, the message area
    resultArea = new TextArea("Please click on Help before you
begin.\n\n",5,50);
    resultArea.setEditable(false);

    resultArea.setFont(font);
    bottomPanel.add("Center", resultArea);

        // Now, add the three panels to the applet with BorderLayout
```

```
    setLayout(new BorderLayout());
    add("North", topPanel);
    add("Center", middlePanel);
    add("South", bottomPanel);

    frame.loadMib();


  } // end of constructor


/** Actions for each of the buttons, choice, and list */
  public boolean action(Event e, Object arg) {

      if (e.target instanceof Button) {   // for each button call a method

           if (arg.equals("ifInOctets")) frame.inOctets();
           else if (arg.equals("ifInUcastPkts")) frame.inUcast();
           else if (arg.equals("ifInNUcastPkts")) frame.inNUcast();
           else if (arg.equals("ifInUnknownProtos")) frame.inUnknown();
           else if (arg.equals("ifInErrors")) frame.inErrors();
           else if (arg.equals("ipInDelivers")) frame.inDelivers();
           else if (arg.equals("ipInHdrErrors")) frame.inHdrErrors();
           else if (arg.equals("ipOutNoRoutes")) frame.outNoRoutes();
           else if (arg.equals("Help")) frame.help();
           else System.err.println("Where did this come from: Button: "
+arg);

      }
      return true;
    }

    /** The start method, used when applet is started
      will not be using it now, since are puting everything in init.
      -- will leave this here for reference */
    public void start() {
//      System.err.println("Starting Sample SNMP Applet");
    }

    /** The stop method, used when applet is no longer on screen */
    public void stop() {
//      System.err.println("Stopping Sample SNMP Applet");
    }

    /** The destroy method needed to stop any threads, etc. */
    public void destroy() {
      System.err.println("Destroying All Sample SNMP Threads");
      if (frame == null) return;
      frame.api.stop();

      // close each session
      for (Enumeration e =
frame.api.sessionList.elements();e.hasMoreElements();)
        ((SnmpSession)e.nextElement()).close();

    }

}

/*****************************************************************************/
/*****************************************************************************/
/* Author:  Christine E. Valliere                        SSN:  585-53-2577  */
/*****************************************************************************/
/*                              Master's Thesis                              */
/*****************************************************************************/
/*     Commitee Members:  Dr. Neilsen, Dr. Mayfield, Dr. Samadzadeh          */
```

```
/*************************************************************************/
/*                DISTRIBUTED REPORTING MANAGEMENT SYSTEM (DRMS)         */
/*************************************************************************/
/*************************************************************************/
/* This file defines the main functions of the DRMS system and calls the  */
/* functions in the Advent Network Management SNMP classes                */
/* This is an extension of sample code provided by Advent Network Mgmt    */
/*************************************************************************/
/*************************************************************************/
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.io.*;
import java.net.*;

import Snmp.*;
import sampleSNMP;
import snmpGraph;

/**
 *  snmpFrame - This is where most of the SNMP operations are done.
 *  Provides a Frame for debug output - which is generated only
 *  when Help is chosen.
 *
 */

public class snmpFrame extends Frame implements SnmpClient {
      // This help/debug window only has a text area and a close button.
  TextArea text = null;
  Button close = null;
  public String oidText = new String(".iso.org.dod.internet.mgmt.mib-2");
  public String mibText = new String("rfc1213-MIB");

      // Need a reference to the applet to get some data
  sampleSNMP browser = null;

      // The fonts used
  Font fontb = new Font("Helvetica", Font.BOLD, 16);
  Font font = new Font("Helvetica", Font.PLAIN, 16);
      // The window title
  String advent = new String("Debug Output (1.0 Beta)");

      // This is the MIB module currently in use - chosen in GUI
  MibModule currentModule;

/** The constructor takes the applet as an argument.
 *  It sets up the window and calls initSnmp()
 */
  public snmpFrame(sampleSNMP  app) {

    initHelpText();

    text = new TextArea(HELPTEXT,30,60);
    text.setEditable(false);
    text.setBackground(Title.fadedGreen);
    setTitle(new String(advent));
    browser = (sampleSNMP) app;
    setLayout(new BorderLayout());
    add("Center", text);
    close = new Button("Close");
    add("South", close);
    resize(700,500);
    setFont(fontb,font);
    initSnmp();

  } // end of constructor
```

```
/** A function for setting the fonts for this window */
  public void setFont(Font fontb, Font font) {

    text.setFont(font);
    close.setFont(fontb);
    setFont(fontb);

  }


/** Event handler for the close button */
  public boolean action(Event e, Object arg) {
       if (e.target instanceof Button) {
            if (arg.equals("Close")) {
             this.hide();
             api.DEBUG = false;
           }
       }
       return true;
  }

/** Need SnmpAPI, SnmpSession, and SnmpPDU instances across many methods */
  SnmpAPI api = null;
  SnmpSession session = null;
  SnmpPDU pdu = null;
  SnmpVarBind varbind = null;

/** To initialize SnmpAPI and SnmpSession classes */
  void initSnmp() {
                  // Instantiate and start SnmpAPI
    api = new SnmpAPI();
    api.start();
    api.client = (SnmpClient) this;

             // Instantiate SnmpSession
    session = new SnmpSession(api);
    session.remote_port = 161;
    session.timeout = 15000;  // 15 seconds
    session.retries = 0;

      // Open the session using applet as argument
      // Will instantiate SASClient and connect to SAServer
      // if needed and feasible
    try { session.open(browser); }
    catch (Exception e) println("Unable to open SnmpSession: " +
e.getMessage());

  }

/** This is the SNMP get request. Create PDU, send, and print result. */
  void getRequest() {
    if (!createPDU(api.GET_REQ_MSG)) return;
    varbind = sendPDU();
    if (varbind != null) println(varbind.toTagString());
    else println("SNMP get request timed out");
  }

/** This is the SNMP get next request. Create PDU, send, and print result. */
  void getNextRequest() {
    if (!createPDU(api.GETNEXT_REQ_MSG)) return;
    varbind = sendPDU();
    if (varbind != null) {
      println(varbind.toTagString());
    } else println("SNMP getnext request timed out");
  }
```

```
/** Function to create PDU with specified command */
  boolean createPDU(byte command) {

            // first get all the options - host and community string
    String opt = browser.host.textfield.getText();
    if (!opt.equals("")) session.peername = opt;
    else {println("No Remote Host Specified");return false;}

    opt = browser.community.textfield.getText();
    if (!opt.equals("")) session.community = opt;
    else {println("No Community String Specified");return false;}

            // Get the OID (Object Identifier)
    String oid_str = null;
//    opt = browser.ltf.textfield.getText();
    oid_str = oidText;

            // Build PDU
    pdu = new SnmpPDU(api);
    pdu.command = command;

            // Create SnmpOID Change any mixed number/name OID to numbers
    if (currentModule != null) oid_str =
currentModule.translateToNumbers(oid_str);
                // instantiate SnmpOID with specified OID
SnmpOID oid = new SnmpOID(oid_str, api);
    if (oid.toValue() == null) {println("Invalid OID"); return false;}

                // add Null valued variable binding with specified OID
    pdu.addNull(oid);
//    println("createPDU, first pdu
element"+pdu.variables.firstElement()+"*\n\n");

    return true;
  } // end of createPDU()


/** Send a PDU synchronously, and check for error returns */
  SnmpVarBind sendPDU() {

    SnmpPDU newpdu=null; // for the received PDU

            // Send PDU
    try { newpdu = session.syncSend(pdu); }
    catch (SnmpException e) println(e.getMessage());

    if (newpdu == null) return null; // timed out

    pdu = newpdu;  // Just like to use the name pdu

    println("Response received from " +pdu.remoteHost+
                        ", community: " + pdu.community);

    if (pdu.errstat != 0) { // error, print.
        println("Error Indication in response: " +
                            SnmpException.exceptionString((byte)pdu.errstat) +
                        "\nErrindex: " + pdu.errindex);
        return null;
    } else return (SnmpVarBind)pdu.variables.firstElement();

  } // end of sendPDU()


/** Print a description of selected MIB node */
  void describe() {
    String opt = oidText;
```

```
    if (opt.equals("")) return;
    if (currentModule == null) return;
    MibNode node = currentModule.getNode(opt);
    if (node != null) println("\n"+node.toTagString()+"\n");
  }


/** create a real-time graph based on polling selected MIB variables
 *  Invokes snmpGraph.
 */
  void graph() {
    if (!createPDU(api.GETNEXT_REQ_MSG)) return;
    element"+pdu.variables.firstElement()+"*\n\n");

    String oidNext=new String(oidText+".1");
    snmpGraph graph = new snmpGraph(this,currentModule,
                        oidNext,session, pdu,browser);
      // Start the graph
    graph.start();
  }

/** The load MIB function */
  void loadMib() {
    MibModule module;
    try {
      URL url = new URL(browser.getDocumentBase(),mibText);
      InputStream stream = url.openStream();
      println("Opened URL: Proceeding with parsing MIB.");
       // Open the MIB module
      module = new MibModule(stream, api,api.DEBUG);
    }
    catch (Exception e) {
      println("Exception: Reading/Parsing MIB URL: " + e);return;
    }
    api.modules.addElement(module);
    currentModule = module;

    if (currentModule == null) {println("No Module Specified");return;}

    String oid = new String(oidText + "." + "");
    MibNode node = currentModule.getNode(oid);

    if (node == null)
      println("Unable to go down specified path in module:
"+currentModule+":"+ oid);

  } // loadMib

  void  inOctets() {
        oidText=".iso.org.dod.internet.mgmt.mib-
2.interfaces.ifTable.ifEntry.ifInOctets";
        graph();
  }

  void  inUcast() {
        oidText=".iso.org.dod.internet.mgmt.mib-
2.interfaces.ifTable.ifEntry.ifInUcastPkts";
        graph();
  }

  void  inNUcast() {
        oidText=".iso.org.dod.internet.mgmt.mib-
2.interfaces.ifTable.ifEntry.ifInNUcastPkts";
        graph();
  }

  void  inUnknown() {
```

```
        oidText=".iso.org.dod.internet.mgmt.mib-
2.interfaces.ifTable.ifEntry.ifInUnknownProtos";
       graph();
  }

  void  inErrors() {
        oidText=".iso.org.dod.internet.mgmt.mib-
2.interfaces.ifTable.ifEntry.ifInErrors";
        graph();
  }

  void  inDelivers() {
        oidText=".iso.org.dod.internet.mgmt.mib-2.ip.ipInDelivers";
        graph();
  }

  void  inHdrErrors() {
        oidText=".iso.org.dod.internet.mgmt.mib-2.ip.ipInHdrErrors";
        graph();
  }

  void  outNoRoutes() {
        oidText=".iso.org.dod.internet.mgmt.mib-2.ip.ipOutNoRoutes";
       graph();
  }

  void help() {
     this.show();
     api.DEBUG=true;
  }

  void println(String s) {
    browser.resultArea.appendText(s + "\n");
  }

  public void debugPrint(String s) {
    text.appendText(s + "\n");
  }
  public boolean authenticate(SnmpPDU pdu, String community) {
      return true;
  }
  public boolean callback(SnmpSession session, SnmpPDU pdu, int reqid) {
    return false;

  }

      // Some helptext for the top of the debug window.
  String HELPTEXT;
  void initHelpText() { // really should be loading this from a URL - maybe
later
    StringBuffer s = new StringBuffer();
    s.append("To view information from a MIB, you must have an SNMP agent
running\n\n");
    s.append("Begin by entering the ip adddress of a host with SNMP -- then
press \n");
    s.append("a button to get the desired information.\n\n");
    s.append("\t\t\t BUTTON DESCRIPTIONS\n");
    s.append("ifInOctets\t\t- Total number of Octets (bytes) received on the
interface, including\n");
    s.append("\t\t  framing characters\n");
    s.append("ifInUcastPkts\t- Number of subnetwork-unicast packets delivered
to a higher-level\n");
    s.append("\t\t  protocol \n");
    s.append("ifInNUcastPkts\t- Number of nonunicast packets delivered to a
higher-layer protocol\n");
    s.append("ifInUnknownProtos\t- Number of inbound packets that were
discarded because of an unknown\n");
```

```
    s.append("\t\t  or unsupported protocol\n");
    s.append("ifInErrors\t\t- Number of inbound packets that contained errors
preventing them from\n");
    s.append("\t\t  being deliverable to a higher-level protocol\n");
    s.append("ipInDelivers\t- Total number of input datagrams succesfully
delivered to IP user\n");
    s.append("\t\t  protocols\n");
    s.append("ipInHdrErrors\t- Number of input datagrams discarded due to
errors in IP header.\n");
    s.append("ipOutNoRoutes\t- Number of IP datagrams discarded because no
route could be found\n\n");
    s.append("If you are loading the MIB browser and SNMP classes locally you
should be\n");
    s.append("able to use SNMP communication and talk to devices without
restrictions.\n\n");
    s.append("Otherwise, it's likely that SNMP operations will not work and a
java application program\n");
    s.append("that is available will be needed on your applet host.\n");
    s.append("MIB loading and browsing will still work with MIB modules in
applet directory.\n\n");
    s.append("rfc1213-MIB, rfc1271-RMON and rfc1155-SMI (an empty module) are
available in\n");
    s.append("the applet directory for loading and testing. Just specify the
name in the dialog.\n");
    s.append("Some of these may also be needed when you have imports in your
other MIB modules.\n\n");
    s.append("DEBUG OUTPUT (On when this window is open):\n");
    HELPTEXT = s.toString();
   }
}

/*****************************************************************************/
/*****************************************************************************/
/* Author:  Christine E. Valliere                          SSN:  585-53-2577*/
/*****************************************************************************/
/*                              Master's Thesis                              */
/*****************************************************************************/
/*        Commitee Members:  Dr. Neilsen, Dr. Mayfield, Dr. Samadzadeh       */
/*****************************************************************************/
/*             Distributed Reporting Management System (DRMS)                */
/*****************************************************************************/
/* This file performs the SNMP get & getNext functions that retrieves a      */
/* point and then updates the graph.                                         */
/*****************************************************************************/
/*****************************************************************************/
import java.applet.*;
import java.awt.*;
import java.util.*;
import java.io.*;
import java.net.*;

import Snmp.*;
import graphDialog;
import snmpFrame;

public class snmpGraph extends Thread {

  snmpFrame frame;
  graphDialog graphd = null;

  SnmpSession session;
  SnmpPDU pdu;
  MibModule currentModule = null;
  int reqid=-1;  // to track different graph instances

  long lastValue=0,valueNow=0,valueHold=467;
  int maxY = 10, maxX = 300;
```

```
int POLL_FREQ = 15000;
SnmpVarBind varbind;

boolean firstPoll = true;

Date date;
long startTime = 0;

public snmpGraph(snmpFrame framein, MibModule module, String oid,
      SnmpSession sessionin, SnmpPDU pduin, Applet app) {
  session = sessionin;
  pdu = pduin;
  frame = framein;


  while (reqid <= 0) { // need a random reqid for each graph
    Random rand = new Random();
    reqid = rand.nextInt();
  }

  pdu.reqid = reqid;

  graphd = new graphDialog(this, app);
  graphd.graph.xMax = maxX;
  graphd.graph.yMax = maxY;
  graphd.graph.Title = new String(session.peername + ": " + oid);
  graphd.graph.xLabel = "Time in seconds";
  graphd.graph.yLabel = "";


  line1 = new Vector();
  Vector data[] = new Vector[1]; data[0] = line1;
  graphd.graph.data = data;

  graphd.show();

}

Vector line1 = null;

public void run() { // the main thread function
  date = new Date();
  startTime = date.getTime();
  frame.println("Starting Graph");
  boolean hold = true;

  while (hold) {
      hold=getPoint(date,startTime);

  }  // end loop forever

} // end of run()

boolean getPoint(Date date, long startTime) {
  SnmpPDU responsePDU=null;

          // Send PDU
    try { responsePDU = session.syncSend(pdu); }
    catch (SnmpException e) { // error sending PDU
     frame.println("Graph: Sending PDU: " + e.getMessage());
       graphd.hide();
       stop();
       return false;
    }

    if (responsePDU == null) { // timeout
        frame.println("Request timed out to: " + session.peername);
```

```
         pdu.reqid++;
          return true;
       }

       if (pdu.errstat != 0) { // error in PDU
          frame.println("Error Indication in response: " +

SnmpException.exceptionString((byte)responsePDU.errstat) +
                        "\nErrindex: " + responsePDU.errindex);
          graphd.hide();
          stop();
          return false;

     } else { // no error in PDU


        varbind = (SnmpVarBind)responsePDU.variables.firstElement();

       Long Val=null; // where we'll store the value
        if (varbind.variable.Type ==SnmpAPI.INTEGER) // if an integer
             Val = new
Long(((Integer)varbind.variable.toValue()).longValue());

        else try Val = (Long) varbind.variable.toValue();  // else try a long
type
       catch (ClassCastException cce) {
         frame.println("Cannot graph the given variable - non-int type
returned");
         graphd.hide();
         stop();
       }

                valueNow = Val.longValue();

        if (!firstPoll)
               updateGraph(valueNow);
        else { firstPoll = false;
              // GET_REQ_MSG = (byte) (ASN_CONTEXT | ASN_CONSTRUCTOR | 0x0)
                 pdu.command = (byte) ((byte) (0x80) | (byte) (0x20) | (byte)
(0x0));
        }
       if (varbind.variable.Type == SnmpAPI.COUNTER) lastValue = valueNow; //
graph diff for counters
       try sleep(POLL_FREQ);
       catch (InterruptedException ie) {}
        responsePDU.command = pdu.command;
       pdu = responsePDU;
       pdu.reqid++;

      } // end else no error

      return true;
  }


  void updateGraph(long valueNow) {

     date = new Date();
     int time = (int)(date.getTime() - startTime)/1000;

     int diff = (int) (valueNow-lastValue);

     if (diff < 0) if (varbind.variable.Type == SnmpAPI.COUNTER)
        diff = (int) (valueNow + 0xFFFFFFFFL - lastValue); // wraparound in 32
bit uns. counter

     double next[] = { (double)time, (double)diff};
```

```
        if ((diff>maxY) || (time>maxX) ) {  // change the graph scales if needed
                if (diff>maxY) {
                maxY = diff;
                graphd.graph.yMax = maxY + maxY/4;
          }
          if (time>maxX) {
                maxX = time+300;
                graphd.graph.xMax = maxX;
          }

            line1.addElement(next);
            graphd.graph.repaint();
        }

        graphd.graph.addPoint(0,next);

    }

}
```

VITA

Christine E. Valliere

Candidate for the Degree of

Master of Science

Thesis: DISTRIBUTED REPORTING MANAGEMENT SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Clovis, New Mexico, On September 8, 1970, daughter of Ralph and Lynelle Gilman.

Education: Graduated from Portales High School, Portales, New Mexico in May 1988; received Bachelor of Science degree in Mathematics from Eastern New Mexico University, Portales, New Mexico in May 1991. Completed the requirements for the Master of Science degree with a major in Computer Scince at Oklahoma State University in July 1996.

Experience: Worked as a systems engineer in the computer and telecommunications department of Conoco and DuPont since June 1991.

Professional Memberships: Desk & Derrick