Preliminary Development of an Inexpensive and Portable Network Monitoring probe using an Internet Embedded Microprocessor

Karthic Rajaram, Montse Ros, Peter James Vial
University of Wollongong

***Abstract*—Management of a telecommunication network involves the requirement to monitor and manage devices on local and wide area networks (for optical, wired and wireless). Devices used to perform this function are referred to as Remote Monitoring devices or network probes. This paper investigates the prototyping of a network probe which uses an embedded internet microprocessor to collect and send data to a network monitor. The device uses the TINI Internet Interface developed by Dallas Semiconductor. This device was chosen as the ethernet driver has a promiscuous mode which potentially allows the capture of all packets sent on the ethernet connection of the embedded internet microprocessor. A prototype system of the Remote Network Monitoring System is developed which utilises Java based software for server and client on a personal computer and some preliminary results are presented. The client software was not implemented on the TINI due to problems associated with memory limitations of the chosen embedded internet processor, with the TINI only having simple functionality of packet capture demonstrated. Further hardware design requirements for the TINI board are needed to provide sufficient resources on the embedded processor to allow for effective packet capture and storage.**

***Index Terms*—RMON probe, TINI, network monitoring, embedded internet device**

I. Introduction

NETWORK Monitoring is the cornerstone of automated network management. The objective of network monitoring is to collect information concerning the status and performance of nodes which are part of a managed network. The International Organisation for Standardisation (ISO) defines the key functions of network management as fault management, accounting management, configuration and name management, performance management and security management. Network monitoring is essentially the backbone which supports the five functional areas of the network monitoring framework.

According to Stallings [1], the information that should be obtained from network monitoring systems can be grouped into three main categories. The first category is Static information that will not change frequently, such as the number of ports on a router. The second category is Dynamic information, which is constantly changing, such as transmission of packets on a network. The final classification is Statistical information which is derived from dynamic information such as packets transmitted by a node to another node in the network. The collection of static information can be made available to the network monitor via a software agent installed in the monitored element. Alternatively it can also be obtained from a proxy agent. Dynamic and statistical information is also collected and stored in a similar manner to static information.

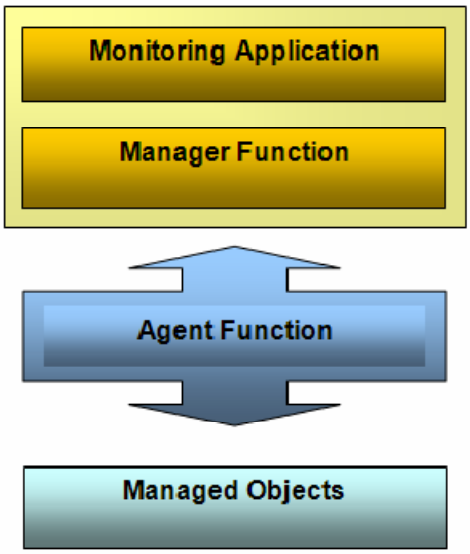

Figure 1: Achitecture of a Network Management System [2]

The architecture of a NMS (Network Management System) can be demonstrated from a functional perspective [2]. This model represents the monitoring system as four functional blocks and is shown in Figure 1. These are the monitoring application, manager function, agent function, and managed objects.

The prototype system used a TINIs400 evaluation board provided by Dallas Semiconductor and integrated available packet capturing libraries such as WinPcap [3] and Jpcap [4] to illustrate how such a system could be potentially deployed in a switched network. Section II includes an examination of

Manuscript received August 1, 2008.
.

the TINI microprocessor used. Section III considers the software used in the PC (Personal Computer) to provide a Remote Network Management System (RNMS). Section IV describes the developed TINI ethernet driver which was able to capture packets, however due to memory limitations it was not able to do this in promiscuous mode and possible solutions to this problem are suggested. Section V describes the software developed for the PC prototyping of the RNMS and provides measured results at different link utilizations. Section VI concludes the paper and provides future areas of development and research.

II. THE TINY INTERNET INTERFACE

A. *TINIs400, TINIm400, DS80C400*

Figure 2 shows the components of the TINIs400 socket and TINIm400 module which was deployed as a portable internet probe in this prototype. The microprocessor used is the DS80C400 designed and manufactured by Dallas Semiconductor to be used in internet embedded applications. It is an 8051 device and has a maximum clock speed of 75MHz. It uses a 24 bit addressing scheme to allow access to up to 16MB of contiguous memory, though the evaluation board only comes with 1MB of SRAM and 1MB of flash RAM. The TINI Operating System is located in the ROM of the DS80C400 microprocessor and this includes the functionality of the IPv4/6 network stack. The network stack provides for a maximum of thirty two concurrent TCP connections with a transfer rate of up to 5Mbps through the Ethernet MAC [5].

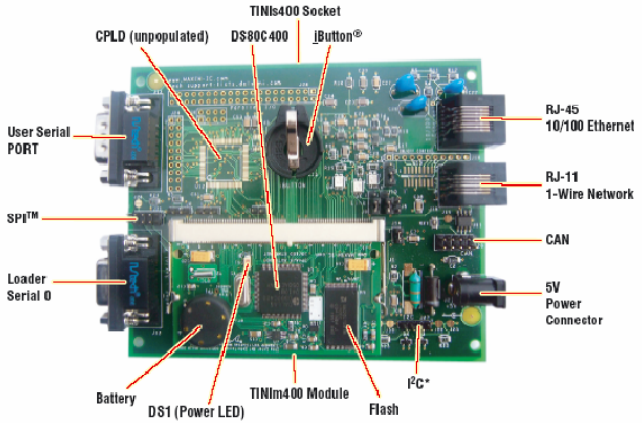

Figure 2: Components of the TINIs400 evaluation board [5]

B. *Packet Capture Role of TINI*

The role of the TINIs400 is to capture the ethernet data packets and send this to the monitoring application which is to be run on a PC via an Ethernet link. The PC would have the analysing modules, the configuration modules and possibly an RMON MIB implemented. As highlighted in Figure 3, the combination of the TINIs400 and the PC (with installed RMON probe software) provides a possible implementation of an inexpensive and highly portable RMON probe.

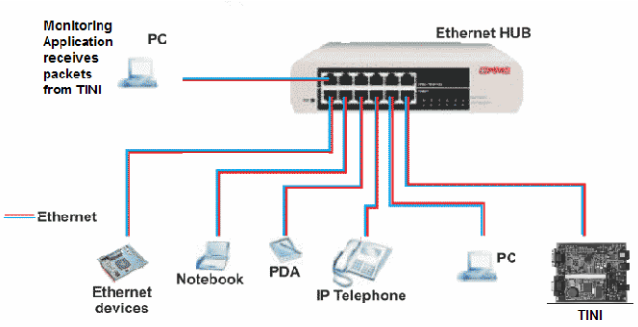

Figure 3: Basic concept of using the TINI and a PC to capture and process packets at the MAC layer

The DS80C400 also has a built-in Ethernet media-access controller (MAC) with an standard media independent interface (MII), the MII defines I/O lines that allow the DS80C400 to communicate with the physical layer interface or network interface card (NIC). The MII contains two basic blocks. The two blocks are the MII I/O Block and the MII Management block. The function of the MII I/O Block is to deliver and receive Ethernet frames to and from an external NIC. Through the MII I/O the DS80C400 is able to support all of the transmit and receive data transactions between the DS80C400 MAC and the external NIC.

The DSTINIm400 is built with the DS80C400 processor and this microcontroller was originally designed with Ethernet networking as its primary objective or application area [6]. With the support of the MII, the DS80C400 allows for the integration of the Ethernet MAC and as a result allows for the capability to access the network stack. Assembly language was used to implement an Ethernet interrupt handler which would be operated in promiscous mode so that all passing packets could be captured by the TINI and resent to the PC. Assembly code was also used to send and receive Ethernet packets to the PC with the monitoring application.

The TINI has a native layer which represents the collection of native methods that support Java's API and provides access to the infrastructure of the TINI Operating System. Between this native method implementations and the interpreted Java code is a very thin layer referred to as the Native Layer Interface. This native method interface is a boundary that must be crossed to switch execution between code being executed by the JVM and a native method. As the TINI RMON project requires native methods, a native library needed to be loaded into the system. Although some support is provided by Dallas semiconductors, the bulk of the library needed to be re-programmed in order to successfully capture the Ethernet packets. Figure 4 shows the relationship between the various layers in the TINI software runtime environment.

The DS80C400 has the ability to capture the MAC layer packets via a special function register (SFR) called the buffer control unit (BCU). The BCU serves as the central controller of all DS80C400 Ethernet activity and is responsible for coordinating and reporting status for all data-packet transactions between the Ethernet MAC and the 8kB dual port

buffer memory [7]. The BCU also regulates CPU read/write activity to the Ethernet controller blocks through a series of SFRs. These SFRs allows the CPU to issue commands to the BCU, exchange packet size/location information with the BCU, configure the on-chip Ethernet MAC, and communicate with external NICs through the MII serial-management bus.

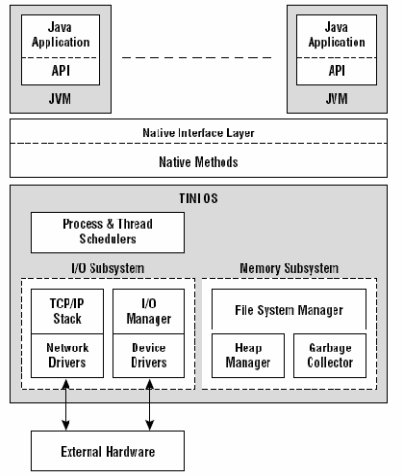

Figure 4: TINI runtime environment [6]

To achieve optimal performance an Interrupt Service Routine (ISR) for handling Ethernet activities should be utilized. The Ethernet ISR can be triggered when the BCU reports the status of either transmit or receive packets [8]. When the ISR is enabled the DS80C400 will be able to capture and send Ethernet packets. The implementation of this ISR is implemented in Assembly and the skeleton for the ISR is provided by Dallas Semiconductors.

III. The Remote Network Monitoring System

We first consider existing network protocol analysers and Remote network monitors in the available research literature and online sources. The objective of this was to assess the functionalities of existing systems to assess the feasibility of incorporating such functionalities in the Remote Network Monitoring System (RNMS). Table I shows the four products which were considered.

A. Ethereal

Ethereal [9] is an open-source, free packet capture tool/packet analyser. Years of considerable development as a protocol analyser have resulted in a feature-rich product with some excellent analysis options. It uses a library known as libpcap to perform the actual packet capture. Libpcap is a system-independent interface for user-level packet capture and provides a portable framework for low-level network monitoring [9]. Libpcap is a free library for developers wanting to write applications for packet capture. The majority of packet capture software interfaces with this library.

B. TCPDump and WinDump

TCPDump is an open-source, command line packet capture tool [3]. It is installed on most Linux systems and is often used by system administrators for a quick overview of network activity. Though a simple application, it has many command line switches which are used to customize its functionality. It again uses libpcap to obtain packets, and though traditionally a Linux application, has been ported to run under Windows and other platforms. WinDump is the Windows version of Tcpdump, WinDump is fully compatible with TCPDump and can be used to watch, diagnose and save to disk network traffic according to various complex rules. WinDump captures using the WinPcap library to capture packets [3].

Product Name	Platform	Capture Method	Interface	Functions
TCPDump [3]	All	libpcap	Command line	Exports Captured Packets to a file
WinDump [3]	Windows	WinPcap (based on libpcap)	Command line	Exports Captured Packets to a file
Ethereal [9]	All	libpcap	Graphical	Protocol analysis tolls, graphing, statistics
RMON-Grabber [10]	Windows /Linux	Proprietary	Graphical	Follows SNMP and RMON 1 standards

Table 1: Existing systems for Network Monitoring

C. RMONGrabber

RMONGrabber is a proprietary closed-source applications marketed at network administrators. Information regarding system design was not available [10]. However the product specifications for this application seem to be similar to that of Ethereal with the exception that the RMONGrabber software follows the SNMP and RMON 1 standards to interact with remote probes[10].

D. Design considerations of RNMS

It became clear that libpcap is the most common method for capturing packets. All but two of the researched products of Table I used this library. The decision was made to use WinPcap which is a windows version of libpcap to capture packets for the portable probe due to the fact it was open source and due to the abundance of support for the module. All of the reviewed products had some method of exporting captured data to a file. The feature of being able to save the captured packets in a common format is essential for the RNMS as it allows the captured data to be analyzed for future use and be used in conjunction with other analysis software. Due to its popularity, libpcap was used as the format for stored capture packets. A feature missing from any of the products which had basic remote capture support was the ability to command more than one capture agent from the client software. Particularly on a larger managed networks where it would be desirable to have agents deployed at strategic points, having to manually keep a list of agent locations would be rather time consuming. One of the major design objectives for the portable TINI probe and management PC was that the developed software should support the ability to control multiple agents concurently from a single management console. Another design consideration was to develop a graphical client with an intuitive and appealing interface, paying particular attention to Human Computer Interface (HCI) principles to ensure an enjoyable user experience. The broad design goals of the RNMS were that the software

needed to interface with WinPcap library using the Java Native Interface. It also needed to establish a means of finding the packet capturing agents in the given network. This was implemented by sending a UDP Broadcast. It also needs to establish a connection between the remote packet capturing agent which is capturing the Ethernet packets and the client which is in essence is the GUI showing the retrieved packet. This was achieved using Java sockets. It also needed the development of a GUI which allows the user to connect to a particular agent, start/stop capturing packets, show the retrieve packets and save the retrieved packets in a pcap formatted file. The design of the Software architecture is provided in Figure 5.

The agent module, shown in Figure 5, should ideally be executed on the TINI board but during development the agent module was executed on a PC. Once the agent is initialized, it creates a Java socket and waits for a client to connect to this socket. When the client wants to find and start retrieving the packets from the agent the client sends a UDP broadcast across the network. The agent replies to this broadcast and hence the agent is identified. Once the agent is identified the client has the option to start retrieving packets from the agent. This is initiated when the client prompts the agent to start capturing packets. The captured packets are sent from the agent to the client via the socket which was established when the agent module was executed. The client GUI is populated by the packets received from the agent in a tabular manner showing the source and destination MAC address of the received packet and the packet type. The client has the option of terminating the packet capture, and saving the retrieved packets into a file for further analysis.

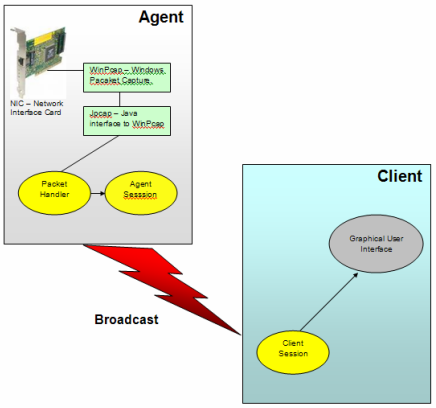

Figure 5: Proposed architecture of RNMS

IV. THE TINI ETHERNET DRIVER

The design of the TINI Ethernet driver is based on the Application note - 712 DS80C400 Ethernet drivers, which were provided by Dallas Semiconductors [11] which provides design considerations for sending and receiving Ethernet packets using the TINI microcontroller. Figure 6 shows the key components of the 10/100Mbps Ethernet controller which is incorporated on the DS80C400. The Buffer Control Register (BCU) is the component which needs to be programmed in developing the Ethernet driver module [7].

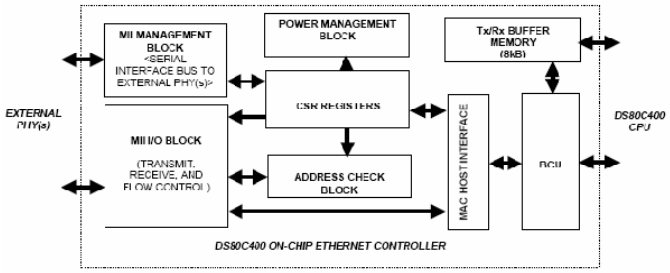

Figure 6: Ethernet controller for TINI [6]

For the purpose of capturing packets it is required that the Ethernet driver has functions to write and read from the specified registers which store the raw Ethernet packets. It is also important to set the MII to operate in promiscuous mode so that all Ethernet packets which pass across the TINIs ethernet interface are captured rather than simply those which are addressed to it (as is normally the functionality of an RMON probe). From the DS80C400's data sheet, the Command/Status Register (CSR) called MAC control register governs the operation characteristics of the DS80C400 in promiscuous mode. Hence using the write function from the Ethernet driver it was possible to write the 18th bit value in the MAC control register to 1; this then forces the DS80400 to operate in promiscuous mode. The design goals of the Ethernet Driver are to set the MAC control register to operate in promiscuous mode; be able to successfully write to the CSRs; be able to successfully read from CSRs; be able to send a Ethernet packet which is placed in the send buffer and finally be able to unload a received packet received from the Ethernet controller.

The Java application which runs on the TINI invokes a native call to the assembly library to set the TINI on promiscuous mode and then start to receive Ethernet packets. These received packets are then sent to the RNMS. In essence the Ethernet driver for the TINI is transparent to the agent module of the Remote Packet Capturing System, however instead of utilizing the WinPcap/Jpcap libraries which was designed for Windows/Linux a library written in assembly is used to capture and send the Ethernet packets. During the design phase of the TINI Ethernet driver it was uncertain as to whether such an implementation was possible on the TINI due to memory and processing limitations. These limitations were discovered during the development phase and are noted next.

Although it was feasible to write the native routine to put the Ethernet interface of the TINI's Operating System in promiscuous mode to get at the raw packet data, the captured packets consumed all of the memory heap quicker than the rate it was possible to read from the memory heap. As a result once the agent application was run on the TINI it crashed the system within a few seconds. The problem lies in the time taken to unload the packet from the buffer memory. The packets which the DS80C400 receives are initially placed in an 8kB Ethernet memory buffer. This memory buffer is divided into receive and send buffer components and is addressed in blocks of 256 bytes. Hence the Ethernet buffer in essence has 32 blocks, which consist of *n* receive blocks and

(32-*n*) send blocks. An Ethernet packet can be anywhere between 64 Bytes to 1600 Bytes in size; in a small network with low traffic the average traffic is in the vicinity of Mb/s. It was thus evident that the buffer memory will not facilitate for operation in promiscuous mode.

An alternate approach of sending the captured packets was also investigated using the serial port. However, the data transfer rate over the serial port was too slow and with the TINI's memory limitation, the same problem was experienced once again. For demonstration purposes a simpler packet capturing routine was written to demonstrate that the Ethernet driver was able to capture packets at the MAC layer. As we could not capture sufficient packets before the buffer overflowed, the current developments indicate that special purpose hardware and memory circuits will need to be designed to do this. This is left as a future activity.

V. PC VERSION OF THE RNMS

A. The Software Used and Modified

It was originally planned during the design phase to develop an interface which used the Java Native Interface to interface between WinPcap which was programmed in C with the agent application written in Java (which would have been installed on the TINI). However after conducting further investigations it was discovered that a wrapper function called Jpcap was already available which performed the interface between WinPcap and Java. Jpcap is a Java class package that allows Java applications to capture and/or send packets to the network. Jpcap is based on libpcap/WinPcap and Raw Socket API. Capturing is performed in promiscuous mode. Following the completion of the Pcap interface, the software was able to successfully capture packets from a local network interface. Jpcap classes were modified to enable them to be serialized and transmitted between the agent and client. This only required adding the "implements Serializable" declaration in the class header, no other changes were made.

There are two general approaches to consider when developing a client/server system in Java. The first is the use of Java Remote Method Invocation (RMI) which is a system developed by Sun to facilitate communication between Java applications over a network connection. The second is to develop a client and server using Java Sockets, and requires developing some form of communication protocol. This is the approach chosen for this project due to convenience, background knowledge in working with Java Sockets and certain disadvantages in using RMI. The disadvantage of using RMI is that it introduces additional overheads (both in the network, and locally maintaining the RMI registry in memory). Using the Sun Java Sockets Tutorial [12] as a basis, a basic client and server was coded. Once this was tested and verified as working correctly, the agent portion was modified to support multiple simultaneous connections, achieved by making it multi-threaded. When a client connects to the agent, on the standard port number (an arbitrary choice of port 4000 is used by default), a new instance of the class AgentThread is created, and from then on the connection is handled within AgentThread. The Agent class, which contains the initial socket for clients to connect to, then returns to waiting for a new connection. In multi-threaded operation each client is assigned a random port number by the operating system. To provide a single facet for communication between the agent and client, a class called ClientSession was developed, which acts as the client-side endpoint of the socket connection. When an agent is added to the client, an instance of ClientSession is created, passing a String representation of the hostname/IP Address of the agent to the constructor. It is usually the case that the client will then invoke ClientSession.connect() method to establish the socket connection. The ClientSession object also provides a send method and this is how the client interacts with the agent. The socket code incorporates the Secure Socket Layer (SSL), a public/private key encryption system. Using the keytool program distributed with the Java JDK, a public/private key pair was generated. With some minor alterations the communications between agent and client were now encrypted. A Graphical interface was constructed using SWT (Standard Widget Toolkit) developed by the Eclipse Platform (which was the SDK used for the PC software developed for the network probe). Figure 7 shows the GUI designed for the PC version of the probe software. Three classes were added to facilitate the automatic discovery of agents within reach of the network. The BeaconThread class was added to the Agent, which runs as a separate thread and opens a socket listening on User Datagram Protocol (UDP) port 4000. BeaconDialog and BeaconSenderThread were added to the Client, the first being the user interface dialog and the second the thread which performs the discovery. BeaconSenderThread sends a 'beacon' UDP broadcast packet to the global address (255.255.255.255) on port 4000. It also opens a UDP socket on the same port to listen for replies. When the UDP socket on the agent receives the broadcast, it extracts the source address and sends back a reply. In this way, it was possible to automatically discover any agents on the network within reach of the global broadcast. BeaconSenderThread also records the time between sending the broadcast and receiving the replies, to produce an estimate of the round-trip latency between the client and remote agents, which is displayed in the dialog.

The next task to be considered was exporting the packet data received by the client to a file, so that it could be analyzed by other software. While researching on the currently available network protocol analyzers and RMON probes it was ascertained that the *libpcap-2.4_little_endian* capture format was widely used for storing lists of captured packets in a single file. The PCapOutputStream follows the Java convention of using OutputStream-like classes for outputting data. To write packets to a file, an instance of PCapOutputStream is created, with a single file argument to the constructor describing the desired file path. The capture file is then ready to be imported to whatever analysis software the user chooses. With the completion of this module, the application could now work in synergy with existing packet analysis tools to provide comprehensive analysis of remote packets. Figure 8 shows the RNMS class structure which has now been described.

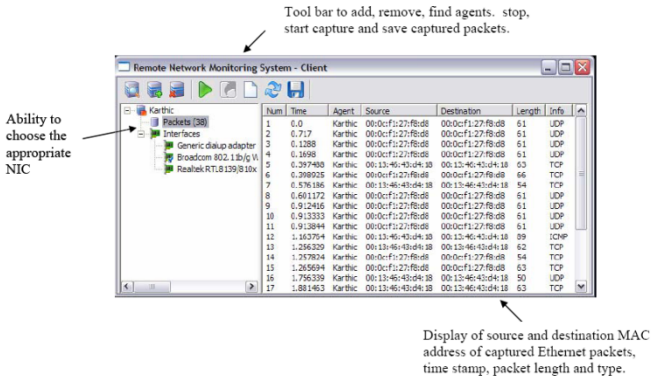

Figure 7: GUI for PC version of probe software

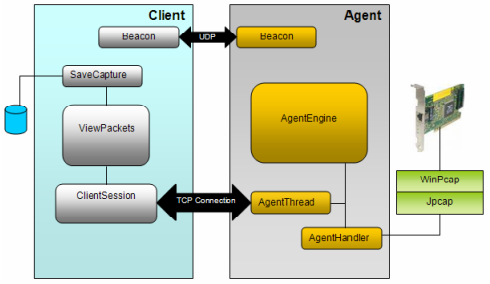

Figure 8: RNMS Class Structure (for PC)

B. *Performance Testing of RNMS PC system*

Performance testing of the RNMS was conducted to ensure it was capable of scaling to large network loads. Two computers were connected via a 100 Mbit switched LAN. One was running the agent program which would output the number of packets received every second onto a console. Another computer was running a packet builder program which was configured to produce packets of 1,400 Bytes length at a user controllable rate. The packet generator used was Pacgen. We used this to adjust the rate of packet production in a controlled way, and examine the number of packets received by the agent. The results are shown in Figure 9, and are based on two test runs over four seconds. Performance was adequate at 25% and 50% levels of utilization, but when the network was fully saturated, the agent processed considerably fewer packets than were being generated. The implementation was analyzed to determine if there were any portions of code which could be optimized to improve this situation. Unfortunately no significant findings were found to pin point the root cause of the difference in packet capture when the network was at at high utilization. We note that the accuracy of the system is satisfactory under low utilization.

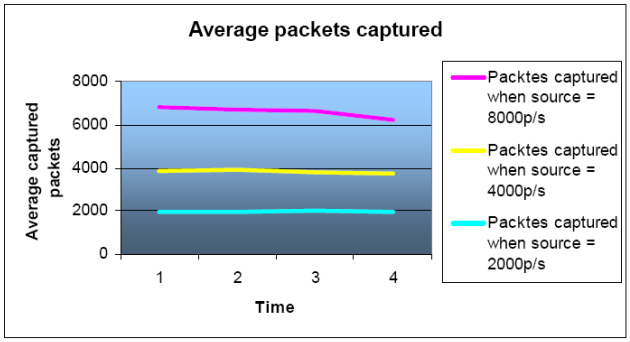

Figure 9: RNMS performance tests

VI. Conclusion

The primary goal of the portable embedded microprocessor, based on the TINI, was to capture Ethernet packets via the TINIs400. This objective was achieved. A PC version of the system was set up and tested. It showed that in a network with low utilization that just over 60% of the traffic could be collected by the probe. This may also be a function of the network interface cards employed and the processing speed of the general purpose computer on which the software was installed (the PC). Future work will involve re-designing the TINI hardware so that packet capture occurs on the TINI and porting the client Java software to the TINI.

References

[1] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2* , 3rd ed., Addison-Wesley, 1996.
[2] D. Chiu and R. Sudama, Network Monitoring Explained: Design and Application, Ellis Horwood, 1992.
[3] WinPcap: The Windows Packet Capture Library", visited 29/08/2006, http://www.winpcap.org/
[4] Jpcap-Java package for packet capture", 29/08/2006, http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html
[5] Maxim/Dallas Semiconductors - DS80C400 Network Microcontroller data sheet, visited 13/06/2006, http://www.maxim-ic.com/products/tini/pdfs/TINIs400.pdf
[6] Maxim/Dallas Semiconductors", visited 10/05/06 http://www.maxim-ic.com/products/tini/pdfs/TINI_GUIDE.pdf
[7] Maxim/Dallas Semiconductors - DS80C400 Network Microcontroller data sheet, visited 13/06/2006, http://www.maxim-ic.com/products/tini/pdfs/DS80C400.pdf
[8] Maxim/Dallas Semiconductors, Appl. Note 712, pp. 1-12.
[9] "SourceForge.net", visited 18/08/2006, http://sourceforge.net/projects/libpcap/
[10] RMONGrabber, visiited 07/29/2008 http://www.wildpackets.com/products/legacy_products/rmongrabber/overview,
[11] Maxim/Dallas Semiconductors, Appl. Note 712, pp. 1-12. 49
[12] "The Java tutorials". Visited 24/08/2006, http://java.sun.com/docs/books/tutorial/networking/index.html
[13] Y.H Choi, K. H. Lee, J. L. Lee and S. B. Lee, "A method of gathering end-to-end management information", Network Operations and Management Symposium, IEEE , Vol. 3, pp. 849-858, Feb 1998.
[14] K. Terplan, *Communication Networks Management*, Prentice-Hall, 1992.
[15] D. Chiu and R. Sudama, Network Monitoring Explained: Design and Application, Ellis Horwood, 1992.
[16] C. M. Kozierok, *The TCP/IP Guide*, No Starch Press, 2005.
[17] S. Waldbusser, Remote Network Monitoring Management Information Base, RFC 2819, 2000.