A Bare PC NAT Box

A.K. Tsetse, A. L. Wijesinha, R. K. Karne, and A. Loukili
Department of Computer & Information Sciences
Towson University
Towson, MD 21252, U.S.A.
{atsetse, awijesinha, rkarne,aloukili}@towson.edu

Abstract—**Bare PC systems are of interest to builders of minimalist platforms in the next-generation Internet. The bare platform enables software to run directly on ordinary PC hardware without using any operating system or kernel. Bare PC systems perform better than conventional systems and are immune to attacks that target the underlying operating system. We have designed and implemented a bare PC system to perform the essential function of NAT (Network Address Translation) that occurs at the boundary of all private and public networks including ISP boundaries in homes and businesses. We compared the performance of the bare PC NAT and that of a Linux-based NAT running on the same hardware in a test LAN environment. The results show that the bare PC NAT has significantly better performance than the Linux NAT with respect to inbound and outbound packet processing time, and throughput, regardless of packet size and payload application type. Moreover, there is a 34% improvement in the maximum number of packets per second (pps) over Linux under heavy traffic. Internal timings on the bare PC NAT box indicate that there is plenty of capacity left for implementing supplementary functions such as packet filtering, deep packet inspection, and routing if needed.**

Keywords-NAT; bare PC; operating system; application object; home router; network security

I. Introduction

Network address translation (NAT) [1] is widely used in the Internet today for both IP address conservation [2] and security. In its most common form, known as Network Address Port Translation (NAPT), NAT translates the private (internal) IP address of the originating host device to a common public IP address representing the NAT device, and the application TCP or UDP port number to an external port number. NAT devices (also known as NAT boxes) are especially critical for corporate and home networks, and are usually co-located with a router and a firewall deployed at the entry/exit points of these networks. NAT functionality can also be provided by security gateways used as tunnel endpoints in IPsec VPNs.

Currently, there is considerable interest in using “minimalist” platforms in the next-generation Internet for a variety of reasons [3]. While such “barebone” systems would likely contain some form of an OS, it is nevertheless useful to consider the extreme case of running software directly on the bare hardware with no intermediary or “layer” in between them. Bare PC prototypes can serve as gateway devices in secure networks due to their intrinsic immunity against typical OS-based attacks. Assuming that IPv4 will continue to exist for a reasonable period of time in the future Internet, it is necessary to implement at least basic NAT functionality on such bare gateway devices.

In this paper, we describe the design and implementation of a self-supporting NAT application that runs on a bare PC with no operating system (OS), and conduct experiments in a LAN environment to compare its performance with a NAT device running on Linux. While it is expected that the bare PC NAT application will have better performance than a Linux NAT due to the elimination of OS overhead, it is important to determine the extent of improvement when designing “barebones” network security devices that incorporate NAT functionality. The remainder of this paper is organized as follows. In Section II, we provide a brief overview of related work. In Section III, we describe the design and implementation of the bare PC NAT application, and in Section IV, we present the experimental results. The conclusion is given in Section V.

II. Related Work

Classically, NAT has been classified into four main types: Full Cone NAT, Restricted Cone NAT, Port Restricted Cone NAT, and Symmetric NAT [4]. However, the most common form used today is referred to as NAPT, which exist in all home and most corporate networks. NAPT requires the translation of both IP addresses and TCP or UDP port numbers in all inbound and outbound packets [1].

Well-known issues that arise due to using NAT are discussed in [5]. For example, to work in the presence of NAT, FTP requires special handling. NAT in general breaks the end-to-end TCP connection paradigm in IP networks [6]. NAT also makes it difficult to initiate direct communication between peers. STUN [7] attempts to address this problem for UDP. In [8], a host-initiated approach to NAT is proposed wherein hosts would perform some of the NAT functions thereby reducing the workload on the NAT device. In [9], a NAT router relays the MAC addresses of internal hosts on a LAN to external hosts enabling them to identify LAN hosts using their MAC addresses. The design, implementation and performance of a programmable network address translator are the focus of [10]. The implementation and performance of a NAT gateway designed for signaling, and NAT traversal for peer-to-peer networks are discussed in [11] and [12] respectively. In contrast to previous work on NAT, this paper describes a novel network address translator that runs as an application on a bare PC with no OS. Such a NAT box is an alternative to a conventional NAT that would run on the hardware (typically supported by some form of an OS-most frequently Linux, or an adaptation of it). More details about bare PC applications are given in [13]-[15].

III. Design and Implementation

The bare PC NAT box runs an application that contains an Ethernet driver, does IP packet processing, and implements NAT functionality. Traffic enters from either of its two interfaces and is forwarded or dropped according to the criteria described below. NAT performance depends on the ability to process packets at high speed and look-up/update the NAT table efficiently when there is heavy traffic and a large number of entries in the table. Furthermore, the NAT application translates the address fields in the IP header and port number fields in the TCP or UDP header of received packets prior to forwarding them. It must also be capable of translating addresses and query IDs in ICMP packets (unless all ICMP traffic is dropped for security reasons).

The software architecture of the bare PC NAT application is shown in Fig. 1. Only two tasks main and receive (RCV) are implemented. These tasks are required by all bare PC applications (additional tasks are added as needed for a given application). In this case, the RCV task is responsible for handling received packets all the way from the Ethernet level through the IP level and NAT-related processing. The main task runs at start-up and whenever the RCV task terminates i.e., after a packet is forwarded.

As in all bare PC applications, the bare PC NAT application is started by invoking an executable boot program stored on a USB flash drive. The initial sector of the USB loads the menus that an administrator can use to select and configure the NAT application. Once the NAT application is selected, a self-supporting bare PC application object containing the monolithic executable is loaded. Control is then passed to the main task. The Ethernet objects ETHOBJ1 and ETHOBJ2 represent the two network interfaces. Depending on the interface from which a packet arrives, the RCV task (RCVTSK in the figure) reads data from either ETHOBJ1 or ETHOBJ2, and a flag is set to indicate the relevant interface. Next, the RCV task calls the NAT Object (NATOBJ) which is responsible for NAT-related processing. The main data structure that stores the NAT table and other relevant protocol-related information is the Transition Control Block (TCB table). Depending on the source and destination IP address of the packet, after translating the headers, the NAT Table is updated accordingly, and the packet is either forwarded to the internal or external interface. We have implemented ETHOBJ1 and ETHOBJ2 as two instances of the same Ethernet object. However, this only works if both network interfaces are to Ethernets.

Fig. 2 shows the main packet processing logic in a bare PC NAT application. Whenever a packet is received, the IP header checksum is verified. If the checksum fails, the packet is discarded. Otherwise, the source and destination IP addresses are checked to determine if the packet entered from the external or internal interface. If the packet is from an interface and destined for the same interface, the packet is dropped. If the packet is from the internal “LAN” interface, NATOBJ checks the TCB table for an existing entry. If an entry exists, its timeout value is refreshed and the translated packet is forwarded to the external interface.

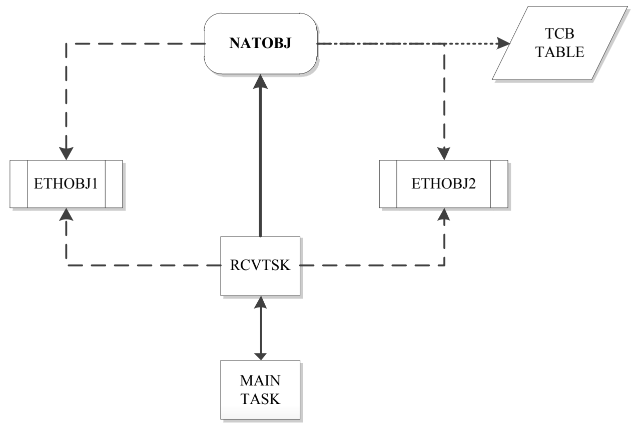

Figure 1. Software architecture of a bare PC NAT

If there is no existing entry in the TCB table, an entry is added prior to forwarding the packet to the external interface. For a packet entering from the external interface, if a valid entry exists in the TCB table, the packet is forwarded to the internal interface; else the packet is dropped. Prior to forwarding a packet, the appropriate IP address and TCP or UDP port number (or ICMP ID) translations are made to the relevant packet header fields. In case ICMP packets are to be handled, the processing logic is similar (except that there are no port numbers).

The AddEntry method was implemented as an overloaded method with different processing for ICMP and TCP/UDP packets. Once this method is called for an outgoing packet that has no prior entry in the table, it invokes the InsertTCB method. This method computes a hash value based on the IP address and port number and returns a pointer to the record number of the next free slot in the table (after doing a search if needed). The record number is the table index and also serves as the translated port (for TCP/UDP) or translated query ID (for ICMP).

Before a record for an outgoing packet is added to the NAT table, the SearchTCB method checks if there is an existing entry in the table that should be used. If so, this method computes a hash value (with a search if needed) and returns a pointer to the corresponding record in the TCB table. For incoming packets (from the external “WAN/Internet” interface), the source IP address and destination port is extracted. Since the destination port serves as an index to the TCB table, a search is not needed to look up the entry in the table. In the case of incoming ICMP packets from the “WAN/Internet” interface, the query ID serves as an index.

Depending on the age field in a NAT entry, it may be deleted from the NAT table.

The TCB table, which is the primary data structure storing all the necessary protocol-related information, can hold a maximum of 100,000 TCB records. Each NAT entry in the TCB table is of type TCB record with the following fields: source and destination IP addresses and ports, protocol, query ID (for ICMP), translated port, translated query ID, age, and availability (which indicates if an active entry is present in a slot). The average size of a NAT entry is 25 bytes. Microsoft Visual Studio was used for developing the bare NAT application. The main code for NATOBJ and other auxiliary protocols were written in C++. Since there is no OS, the application does not use any system calls. The drivers for the hardware interfaces (NIC, keyboard, display, and USB) were written using a combination of assembly and C code.

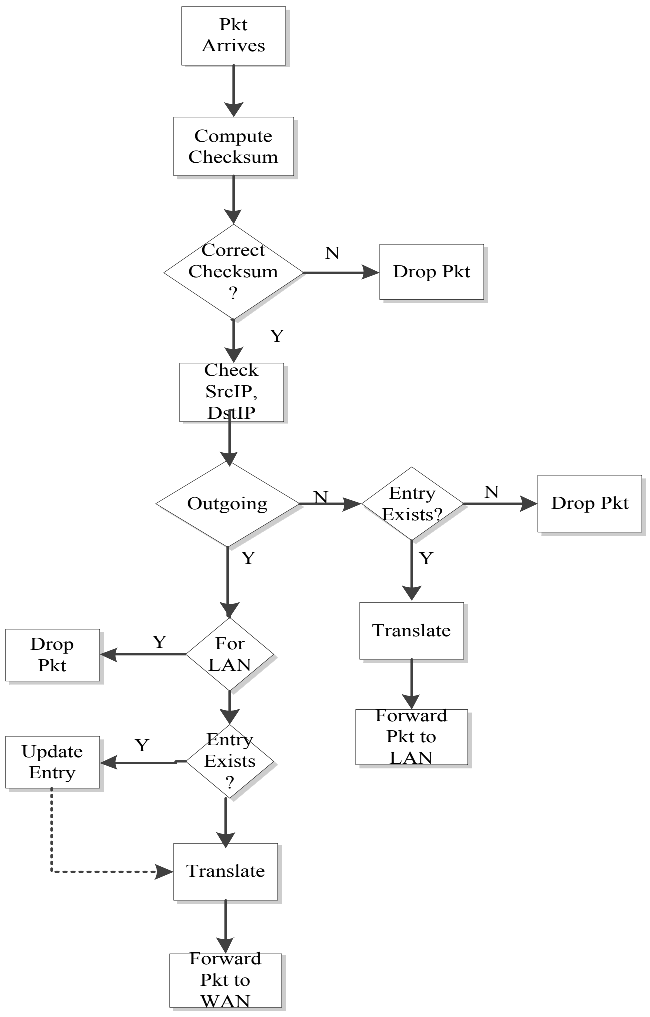

Figure 2. Packet processing logic

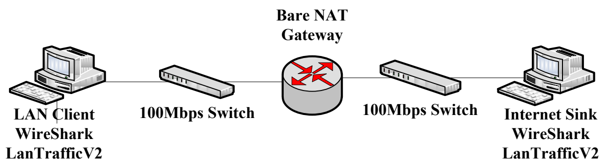
Bare NAT Gateway
LAN Client WireShark LanTrafficV2
100Mbps Switch
100Mbps Switch
Internet Sink WireShark LanTrafficV2

Figure 3. Test LAN

IV. Experimental Results

The test LAN used for the performance study is shown in Fig. 3. All links and the two Ethernet switches were 100 Mbps and the NAT boxes to be tested were placed between the switches. Although the figure only shows a single client, more clients were added as needed. We conducted experiments to study NAT performance under both normal and heavy traffic. Three different configurations were used: no NAT, bare PC NAT, and Linux (Fedora) NAT. The detailed hardware and software specifications are given in Table I. Three types of traffic (TCP, ICMP, and HTTP) were used to generate normal (moderate) loads as follows: first, the source clients were configured to establish 16 simultaneous connections via LanTraffic V2 [16] to the sink using different ports and different IP datagram sizes. These sizes were based on the guidelines given in [17]. For each size, the source node then continuously transmitted TCP data to the sink for 10 minutes and the data was collected using a Wireshark packet analyzer and the LanTrafficV2 network tool. For ICMP, the ping command was used to directly transmit 200 packets between source and the sink for each datagram size. For HTTP traffic, http_load [18] was used to request files of varying sizes from an IIS Web server running on Windows 2008/server. To ensure fair comparison between the bare PC and the Linux (Fedora 12) NAT, all non-essential services on the latter were disabled with the exception of IPtables.

TABLE I. Hardware and Software Specifications

PC/Switch	**Role**	**Hardware**	**OS/Software (non-bare only)**
Dell Optiplex GX270	bare/ Linux NAT Gateway; bare client	Pentium 4, 2.4 GHz, 512 MB RAM, 3Com 10/100 NIC, Intel PRO/1000	Fedora Core 12 (2.6.18-92.1.22) IP Tables
Dell Optiplex GX755	Web server sink; OS clients	Pentium 4, 2.4 GHz, 1GB RAM, Intel PRO/1000 NIC	Windows 2008 Server, Fedora Core 12 (2.6.18-92.1.22) http_load, LanTraffic V2, Windows XP SP3, Mgen5
Netgear GS108	Ethernet Switch		

To compare performance of the NAT boxes under heavy traffic, a) a mix of UDP and TCP traffic (80% TCP and 20% UDP) and b) TCP traffic only was generated.

For this purpose, bare PC TCP clients and Windows UDP clients running Mgen5 [19] were used as sources; and an IIS Web server running on Windows 2008/server (configured to have 10 different IP addresses) was used as the sink. The bare clients established TCP connections to the server using different source/destination IP addresses and port numbers.

A. Throughput and RTT

Fig. 4 compares the throughput for TCP traffic with and without NAT using LAN_Traffic V2. As expected, throughput without using NAT is higher compared to that using a bare PC/Linux NAT. However, the peak throughput is about 70 Mbps for the bare PC NAT and about 63 Mbps for Linux (i.e., an improvement of about 11%). Fig. 6 compares the round trip time (RTT) for ICMP traffic, which reflects the inbound and outbound NAT processing time. It can be seen that the processing times for the bare PC NAT are considerably less than those for the Linux NAT.

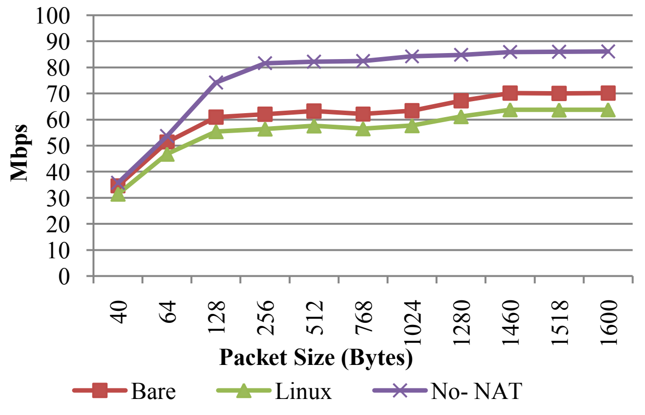

Figure 4. Throughput

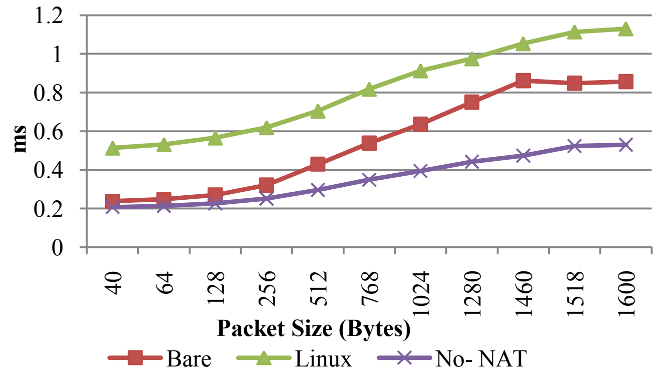

Figure 5. RTT (ICMP)

B. Connection and Response Times

To measure connection and response time, HTTP_load was used to generate requests. In Fig. 6, the connection time for the Linux NAT increases with increasing file size to a peak value of 0.262ms for a size of 32KB. For the bare PC NAT, a relatively constant connection time of 0.214ms is maintained irrespective of the file size since data copying is minimized. In Fig. 7, the response times for the Linux and bare PC NAT are 0.522ms and 0.5ms respectively. In both cases, a small performance improvement of about 5% is attained by the bare PC NAT.

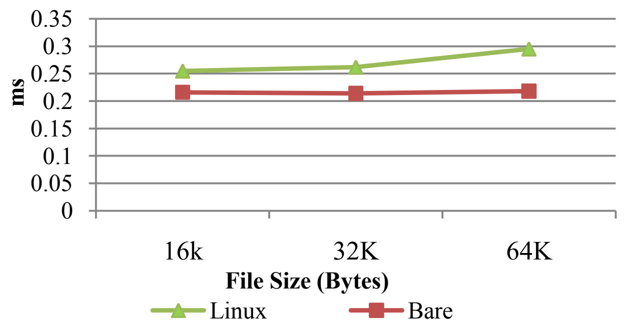

Figure 6. Connection time

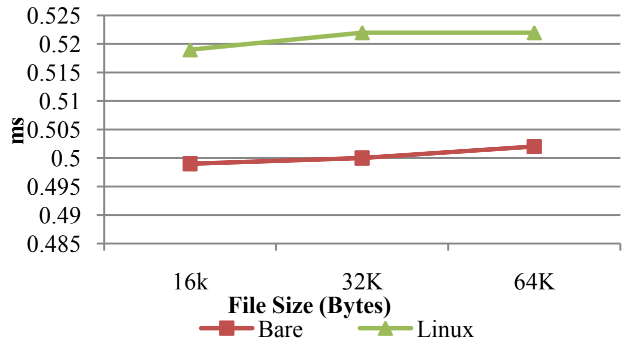

Figure 7. Response time

C. Packets Processed Per Second (PPS)

To measure pps, bare clients were used to generate TCP traffic. In Fig. 8, it is seen that the maximum number of packets processed per second (pps) by the Linux and bare NAT for TCP traffic only are 9,560 pps and 13,992 pps respectively (i.e., a 46% improvement). In Fig. 9, the corresponding maximum pps values for a traffic mix of UDP and TCP generated using bare clients and Mgen5 are 9,522 and 12,735 for the Linux and bare PC NAT respectively (i.e., a 34% improvement).

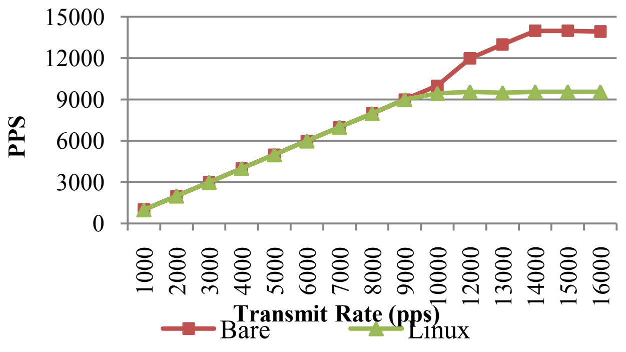

Figure 8. Packets processed per second (TCP)

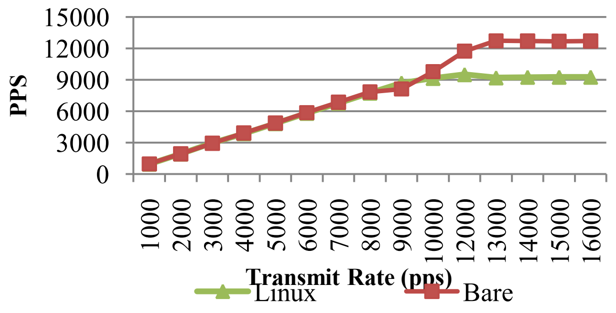

Figure 9. Packets processed per second (traffic mix)

D. *Internal Timings*

Internal timings for key operations on the bare PC NAT are shown in Table II. The table also shows that the total CPU utilization measured at a maximum of 14,000 pps is 0.46%. This implies that the unused CPU cycles could be used by other processes (for example, to improve NAT security). Packet forwarding is seen to be the most expensive operation. This is because (unlike the *Lookup* and *AddEntry* functions, which only process headers), the forwarding function has to take into account the payload as well. The *AddEntry* function for outgoing packets is more expensive than *Lookup* because adding an entry to the NAT table requires a prior search and computation of a hash value.

TABLE II. INTERNAL TIMINGS FOR THE BARE PC NAT

Function	**Metric**	
	Processing Time (µs)	CPU Utilization (%)
Add Entry	1.83	0.144
Look Up	0.165	0.12
Forwarding	1.86	0.196

V. CONCLUSION

We described the design and implementation of a NAT application that runs on a bare PC with no OS. We also compared the performance of the bare PC NAT with a Linux NAT. The bare PC NAT has less overhead than the Linux NAT and performs better with respect to throughput, inbound and outbound processing, and packets processed per second. Results for moderate and heavy traffic, and for different packet sizes and various traffic types, were presented. Internal timings showed that while packet forwarding is the most expensive operation on the bare PC NAT, there is sufficient capacity left for implementing additional functionality such as security-related processing and routing. Bare PC NAT applications could be used in the next-generation Internet to enhance performance as well as security.

REFERENCES

[1] P. Srisuresh and K. Egevang, "Traditional IP network address translator (Traditional NAT)", RFC 3022, IETF Jan 2001.

[2] M. Boucadair et al., "Anticipate IPv4 address exhaustion a critical challenge for internet survival", 1st International conference on evolving Internet , pp. 27-32, 2009.

[3] S. Sen, R. Guerin and K. Hosanagar, "Functionality-rich versus minimalist platforms", Computer Communication Review, 41(5), pp. 36-43, 2011.

[4] J. Rosenberg, J. Weinberger, C. Huitema and R. Mahy, "STUN – Simple traversal of user datagram protocol (UDP) through network address translators (NATs)", RFC 3489, IETF Mar 2003.

[5] T. Hain, "Architectural implications of NAT", RFC 2933, Nov 2000.

[6] S. Guha and P. Francis, "Characterization and measurement of TCP Traversal through NATs and firewalls", Proceedings of the 5th ACM SIGCOMM conference on Internet measurement (IMC '05).

[7] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session traversal utilities for NAT (STUN)", RFC 5389, Cisco October 2008.

[8] L. Zheng, "Host-initiated NAT, " IETF Draft, March 2001.

[9] R. Murakami, N. Yamai and K. Okayama, "A MAC-address relaying NAT router for PC identification from outside of a LAN", 10th Annual international symposium on applications and the Internet, pp. 237-240, 2010.

[10] T-C. Huang, S. Zeadally, N. Chilamkurti and C-K. Shieh, "A programmable network address translator: design, implementation, and performance", ACM Transactions on Internet technology, Vol. 10, No. 1, Article 3, February 2010.

[11] R. Bless and M. Rohricht, "Implementation and evaluation of a NAT-Gateway for the general Internet signaling transport protocol", IEEE 12th International conference on high performance computing and communications, September 2010.

[12] Z. Xiangming and W. Zheng, "A NAT traversal mechanism for peer-to-peer networks", International symposium on intelligent ubiquitous computing and education, 2009.

[13] L. He, R. Karne and A. Wijesinha, "The design and performance of a bare PC Web Server", International Journal of Computers and their Applications, vol. 15, June 2008, pp. 100 - 112.

[14] G. Ford, R. Karne, A. L. Wijesinha and P. Appiah-Kubi, "The design and implementation of a bare PC email server", 33rd Annual IEEE International computer software and applications conference (COMPSAC), 2009.

[15] G. H. Khaksari, A. L. Wijesinha, R. K. Karne, L. He and S. Girumala, "A peer-to-peer bare PC VoIP application", Consumer Communications & Networking Conference (CCNC), pp. 803-807, Jan 2007.

[16] LanTrafficV2 , http://www.zti-telecom.com , accessed 09/20/2011.

[17] S. Bradner and J. McQuaid, "Benchmarking methodology for network interconnect devices", RFC 2544, IETF Jan 2001.

[18] http_ load, http://acme.com, accessed 09/20/2011.

[19] Mgen, http://cs.itd.nrl.navy.mil, accessed 09/20/2011.