A New Scalable Framework for Emulating Huge Networks

M. Zahid
Research Laboratory STRS
INPT, Av. Alla Al Fassi – Madinat AL Irfane
Rabat - Maroc

A. Mezrioui
Research Laboratory STRS
INPT, Av. Alla Al Fassi – Madinat AL Irfane
Rabat - Maroc

A. Belmekki
Research Laboratory STRS
INPT, Av. Alla Al Fassi – Madinat AL Irfane
Rabat - Maroc

ABSTRACT

Network emulation and simulation tools are widely used for preproduction, studies and researches purposes. This success is due to the quality of result they provide compared to the real equipments. Another advantage of the network simulators is that the cost of studies and experiences are exponentially reduced especially for networks that use expensive hardware or a big number of nodes. In the work done for fighting against Distributed Denial of Service (DDoS) based on Botnet (malicious programs that take the control of many machines on behalf of the owners in order to attack services or send spams), a real time test of the trace-back and counter-attack algorithms is needed. So the emulation tool should be scalable in order to create thousands of bots with fewer resources. Unfortunately, the existing emulation/simulation tools suffer from some limitations like the nodes number that cannot exceed hundreds, have simulation concept or not scalable. That's why it was decided to develope a new network emulator that implements huge networks. The aim of this paper is to present this new scalable framework that help to emulate network equipments and application based on UDP and TCP protocols with a huge number of nodes. It was developed basically for emulating DDoS attacks based on Botnets but it can be used for any other purposes like stress test for HTTP servers or telephony over IP services etc.

General Terms

Computer science, computer and network emulation, computer security

Keywords

Simulation; Emulation; Network emulation; Network security; Botnet; TCP/IP; DDos; BrutForcing; Virtualization; GNS3; VMWare; VirtualBox; Qemu

1. INTRODUCTION

Distributed Denial of Service (DDoS) based on Botnet is one of the most dangerous and widely used attacks in the internet [1] [2]. This kind of attacks needs a hundred thousands of bots in order to generate a huge number of requests or traffic and then make the target service unreachable using the techniques like the Internet Control Message Protocol (ICMP) flood, the SYN flood, R-U-Dead-Yet (RUDY) attack and slow Read attack [3].

For studying the Botnet attacks behavior and fight against them, researchers need huge resources in terms of machines and network equipments in order to establish the real word conditions. Those conditions can help when testing algorithms like detections and trace-back ones [4] [5]. For that, five basic requirements that emulation/simulation tool should provide was defined:

- Ability to create thousands of nodes
- Being based on emulation concept.
- Ability to consume fewer resources in terms of CPU and memory
- Scalability and the ability to add new protocol and network nodes easily.
- Ability to be distributed on several machines.

Because the existing tools can't meet these requirements, a new framework that contains two components was developed:

- Network emulator which is a tool that emulates the network links and equipments like Ethernet links, switches and routers.
- Lightweight C++ Virtual Machines where was developed a network module that provides basically Ethernet, ARP, IP, ICMP, UDP, TCP and DNS protocols so they can execute any application based on those protocols through the Network emulator. These machines contain open interfaces that allow adding new protocols (FTP, HTTP, SIP, RTP …) easily. They can also reach 10000 operating virtual machines over a real one.

The idea behind this framework is to provide an open test platform that can be used for any kind of test related to the computer network or network based software. For that, it was divided into scalable modules for both network equipments and protocols. This framework is shared as an Open Source project under GNU General Public License version 2.0 (GPLv2) in source forge web site.

The rest of this paper is structured as follows. Section II explains difference between simulation and emulation tools and introduces the reason behind proposing this framework. Section III describes the different modules and connectors of this framework. Section IV then describes the architecture and explains how to use it by providing some code samples. Section V presents the test procedures and results. Finally, section VI show the future works.

2. BACKGROUND AND MOTIVATION

This section explains the existing test methods and the reasons behind developing such framework.

2.1 The existing test methods

Many researches were done for detecting DDoS or fighting against them by providing new architectures and algorithms like probabilistic packet marking algorithm that allows detecting the attack source [4] and the architecture that detects the DDoS/Brute forcing attacks for destroying the Botnet behind [5]. However, researchers have to implement a big network with a big machines number in order to test their proposals. So they often use the following techniques:

- Using log files from an Internet Service Provider (ISP): This technique is based on the use of ISP log from a detected DDoS traffic. These files are taken from back office ISP equipments suspected to be bridge for DDoS attacks.
- Using simulation tools: This technique is based on implementing both the model of a Botnet and the proposed algorithm or architecture in the simulation tool. Many simulation tools provide protocols used by the Botnet like IRC and peer-to-peer [6].
- Using virtual machines inside a super calculator: This technique consist of running a big number of virtual machines (1 million machines according to Sandia National Laboratories) inside a super calculator and installing bots inside them [7].

Every techniques of those listed below has an inconvenient:

- The log files used for test may not contain all the attack data so the efficiency of the proposed solution cannot be determined correctly.
- The integration of an algorithm seems to be an easy task in a simulation tool. But when we want to test a distributed architecture the integration become complicated especially if the used algorithms are based on packet content analyses. Furthermore, the simulation tools can't provide a very big number of nodes.
- The use of virtual machines consumes a lot of resources and money so it's not accessible for everybody (US$ 100,000 for creating 1 million machines according to Sandia National Laboratories) [7].

2.2 Choosing between simulation and emulation concepts

The simulation technique is very useful since it can model the network or protocol behavior by calculating the interaction between the different network components. These kinds of tools are called discrete event simulators because they use mathematical formulas as functions of time. These formulas help to calculate the nodes or protocols behavior using a given period.

The emulation technique consists of programming a container that has the same characteristics as a device or network layer. This container can then be used to execute the real operating system or encapsulate a real traffic. By that it emulates device or network protocol behaves like it was in real world.

The aim behind this framework is to provide a tool that can interact with real/virtual equipments and applications in real time. So emulation concept seems to be the most adequate for this purpose.

3. THE NEWPROPOSED FRAMEWORK

This framework meets the requirements defined previously and it is divided into two components. The first one emulates the network equipments and links; the second one provides the IP protocol stack for emulating a huge number of machines.

3.1 Network Emulator component

The Network Emulation is designed to be a new tool for emulating networks that handles a huge traffic. Unlike Dynamips (GNS3) that can handle only 1000 packets per second [8], this component can manage a higher packet rate and provide similar connecting interfaces. It is basically made for research purpose and especially for studying and testing models that contains a very big number of nodes such as DDoS attacks based on Botnet IRC and peer2peer.

The Network emulator component is a distributed test environment that can be deployed over many machines. It provides network equipments like routers and switches; it provides also connectors to the external applications (Virtual machine, Lightweight C++ Virtual Machine, VPCS) and the real networks.

3.2 Lightweight C++ Virtual Machine (LwCVM) component

The Lightweight C++ Virtual Machine is written in C++ that allows creating thousands of virtual machines in the same physical one. In these machines, only the TCP/IP stack is implemented in order to be as light as possible and use limited resources in terms of Central Processing Unit (CPU) rate and memory allocation. Those machines can also execute a network based C++ code.

This component is based on the Open Source project lightweight IP (LwIP) [9] which is a lightweight and independent implementation of TCP/IP stack. This project provides a portable source code written in C programming language with many features for all the TCP/IP layers. These features depend on the different layers like checksum (CRC) calculators and verifiers, packet generators, memory manager, packet field analyzers and generators etc. This project was created in order to provide a lightweight TCP/IP stack for embedded systems with limited memory and CPU frequency.

4. Architecture of the framework

This session explains the architecture of both components: Network Emulator and LwCVM.

4.1 The Network Emulator architecture

The Network Emulator modeling is based on the Object concept in order to benefit from the Object Oriented Programming (POO) advantages which are making the program structure clear and modular, easy to maintain and also scalable for adding new features. The following figures illustrate the architecture of this component.

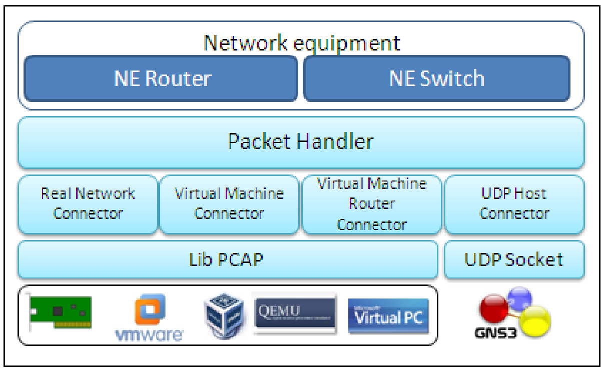

Fig 1: Network emulator architecture

The Network Emulator provides the following network equipments (see figure 1):

- Network Emulator Switch (NE Switch): it represents a network switch that allows connecting machines to each others. The main role of this equipment is to redirect the Ethernet packet according to the destination hardware address.

- Network Emulator Router (NE Router): it represents a network router that allows connection between different networks. The role of this equipment is to redirect packet according to the destination IP address.
- The Network emulator provides also some external connectors:
- UDP Host: it allows connecting the Network emulator to LwCVM, Virtual PC Simulator (VPCS) [10], Network Emulator running over another machine and GNS3. This connector is based on UDP protocol. It uses two UDP ports. The first one is used for receiving UDP packet, the second one is used for sending UDP packet to the remote host. The UDP protocol is used here to encapsulate the Ethernet packets generated by the LwCVM, VPCS, Network Emulator or GNS3.
- Virtual Machine Host: it allows connecting the Network emulator to a virtual machine (VMware, VirtualBox, Qemu …) through a virtual network card. So this last can be considered as a part of the created network and can interact with the other equipments transparently.
- Virtual Machine Router: it allows connecting the Network Emulator to a virtual machine based on Linux Micro Core. We have installed in this machine all the services we can find in a real network router like NAT, Firewall, routing protocols (RIP) etc.
- Real Network Connector: it allows connecting the Network Emulator to the real network through the physical machine network card. So the virtual machines and network equipments that can interact with the real network transparently.
- The last three connectors are based on PCAP library that allows access to the real and virtual network cards.
- The Packet handler module ensures the transfer of the Ethernet packet between the network equipment and the external connectors. This handler is based on a thread safe FIFO queue.

4.2 The LwCVM architecture

The LwCVM component was designed according to TCP/IP model (see figure 2). So every component represents a TCP/IP stack layer starting from the Ethernet layer to the TCP and UDP ones.

Every layer can only communicates with the direct upper one in order to handle the received packets and the direct lower one in order to encapsulate the send ones.

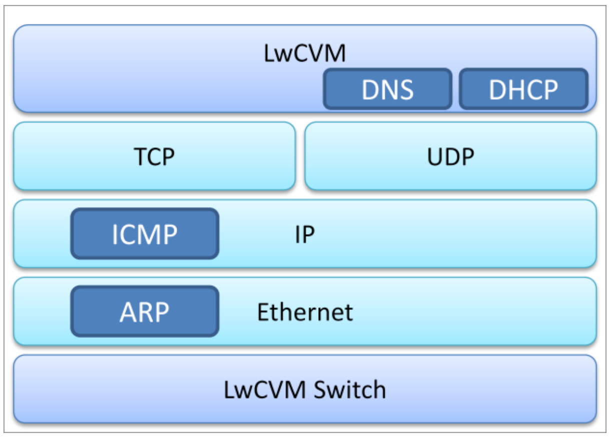

Fig 2: LwCVM architecture

All the methods that allow the TCP/UDP socket management in C standard library like **createSocket()**, **bind()**, **connect()**, **setSocketOption()** and **closeSocket()** are implemented in the class LwCVM that represents the lightweight virtual machine. This class provides a virtual method **main()** that developers can redefine and implement their socket based program. This method is a self threading, so it can be executed in a separate thread only by calling the method **start()**.

This framework can be connected to the Network Emulator framework through an UDP connection. But if we want to use a huge number of LwCVM we will need a double number of UDP Ports. That’s why we introduced a new component LwCVMSwitch that represents a switch. The role of this switch is to aggregate many LwCVM in the same UDP port. By that we can decrease significantly the number of used UDP ports in the physical machine.

4.3 Adaptation of LwIP source code

In order to use LwIP source code and adapt it, several changes have been done:

- The use of object-oriented programming language instead of procedural programming in order to organize the code according to the different layers.
- The use of pThread API [11] to manage threads. The TCP/IP stack needs timers in order to correct packet lost and errors. Furthermore, the new API provides TCP and UDP sockets which are a thread blocking methods. Finally, the pThread API is compatible with both Linux and Windows.
- The adaptation of the link layer to our context. This adaptation concerns the low level so we changed the methods that communicate with the network card by new ones that encapsulate the Ethernet packet in UDP sockets.
- Every equipment/connector in the Network emulator has a separate thread that handle the received packet. That’s why we use a safe threading FIFO queue in order to exchange packets between them.

4.4 How to use this framework

As we have seen, this framework is divided into two components. So the use of this framework can be done in two steps.

4.4.1 How to use Network Emulator

This example explains how to create a simple network that contains two machines (Virtual machine and LwCVM one) using the Network Emulator (see figure 3):

1. Instantiate two connectors: virtual machine and UDP Host connectors, these connectors will be used to connect respectively a virtual machine and a LwCVM to our network.
2. Instantiate a NE Switch
3. Connect the NE Switch to the connectors created before.
4. Prepare and launch a LwCVM (see section IV.D.2).
5. Prepare and launch a virtual machine using VirtualBox, VM ware or Qemu
6. Start the NE switch and the connectors.

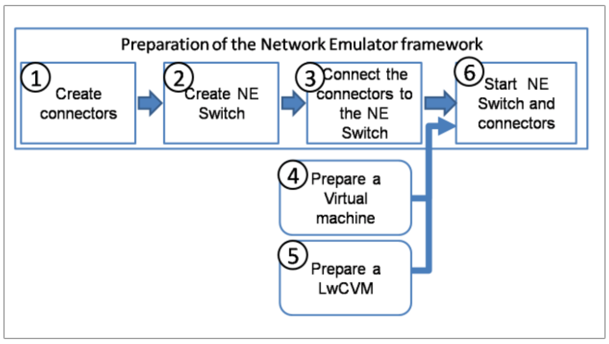
Preparation of the Network Emulator framework
1 Create connectors
2 Create NE Switch
3 Connect the connectors to the NE Switch
6 Start NE Switch and connectors
4 Prepare a Virtual machine
5 Prepare a LwCVM

Figure 3: Network emulation steps

4.4.2 How to use LwCVM

This part presents how to use the LwCVM in order to create a lightweight virtual machine that sends ping request every second to another machine. This manipulation has to be done in two steps:

1. The first step consists of creating a subclass of the class LwCVM and redefine the method main(). In this method we can develop the program of the emulation by using the available socket methods. In this example we will send a ping request to a virtual machine with the IP address 192.168.1.11 (see figure 4).
2. The second step consists of instantiating the previous subclass, setting its IP address and a network mask and connecting it to the switch. This last will ensure the connection with the Network Emulator through the UPD ports 30100 and 20100 for sending and receiving the encapsulated Ethernet packets (see figure 5).

```
// create a subclass of the class LwCVM
class myLwCVM: public LwCVM
{
public:
// redefine the method main
        int main()
        {               while(true)
                {
/*sending a ping request to the host 192.168.1.11every
second*/
                        sendPingRequest(" 192.168.1.11");
                        usleep(1000000);
                }
        }
};
```

Figure 4: Redifinition of the LwCVM behavior

```
//instanciate the switch class
LwCVMSwitch *lSwitch= new LwCVMSwitch ();

/*initiate the switch to be able to connect Network Emulator
through UDP protocol using the following parameter: local port,
remote port and host*/
lSwitch->init_connection(30100,20100,"localhost");

// instanciate the lightweight virtual machine
myLwCVM *myVM= new myLwCVM ();

/*initiate the LwCVM with the parameters: IP address, network
mask, gateway IP address, MAC address*/
myVM ->init("192.168.1.10"," 255.255.255.0" ," 192.168.1.1");

//connect the LwVM to the switch
lSwitch ->connect(myVM);

//start the switch and the LwCVM
lSwitch ->start();
```

Figure 5: Instantiation of the LwCVM

5. TEST OF THE LWCVM AND NETWORK EMULATOR

This section describes the test scenarios done using this framework. The tests are divided into two parts:

- The first part concerns the framework performances.
- The second one concerns a DDoS attack test based on ping requests.

5.1 Test environment and constraints

The entire tests were done using a machine with the following characteristics:

- Linux 32 operating system
- Microprocessor Core 2 Duo 2.1 Ghz
- 4 Gb of RAM.
- 2 Mb of cash memory

Because the default thread stack size is 8 Mbyte. It was not possible to instantiate more than 200 LwCVM in the same process (every LwCVM needs two threads. The first one manages the received packet and the second one executes the main method). To deal with this limitation, the stack size was decreased to 256 kbyte using the command “ulimit -s 256” so it was possible to instantiate up to 3000 LwCVM per process.

5.2 Performance test

5.2.1 Latency test and packet lost ratio

5.2.1.1 Tests description

The aim of this test case is to calculate the network latency over the Network Emulator. For that, we created two networks using a router (NERouter) and two switches (NESwitch). Each network contains the same number of LwCVM that send 5000 ping requests to the opposite one with a period of 10 ms.

During the test, the number of LwCVM was increased in each network and the latency values were saved in a text file. Then we calculate the average latency for all the machines.

The following figure (6) illustrates the latency test.

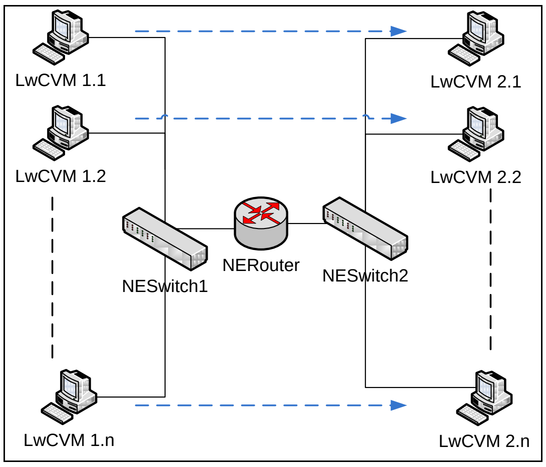

Figure 6: Network Emulator latency test

5.2.1.2 Test result

The following chart (figure 7) illustrates the latency value in millisecond for the ping test. It is evident that this frameworks provide a good latency even with 2000 nodes (less than 2.5 ms).

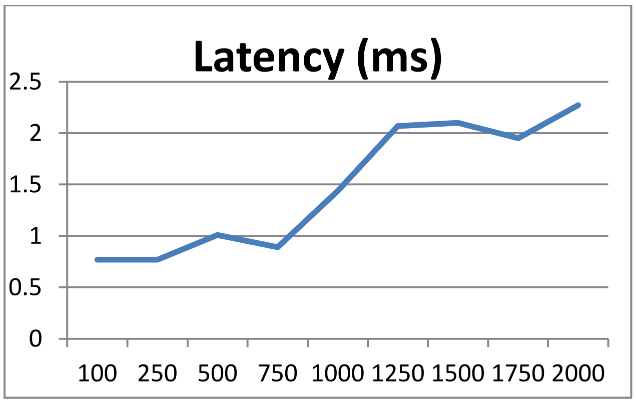

Figure 7: Latency test result

The following chart (figure 8) illustrates the packet lost ratio for the ping test. It shows that this framework provide a low packet lost ratio with more than 2000 connected nodes.

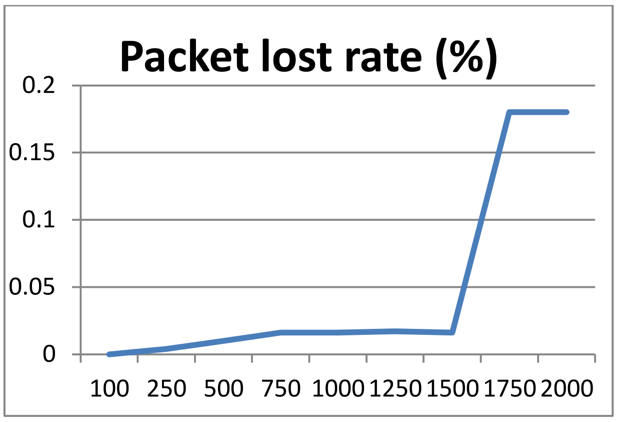

Figure 8: Packet lost rate test result

5.2.2 Memory test

5.2.2.1 Test description

The aim of this test case is to calculate the memory consumed by the LwCVM during the program execution. To reach this aim, the number of instantiated LwCVM was increased progressively.

5.2.2.2 Memory test result

The following chart (figure 9) illustrates the memory used by the LwCVMs and shows that this framework consumes less than 25 Mb of memory with 1000 machines.

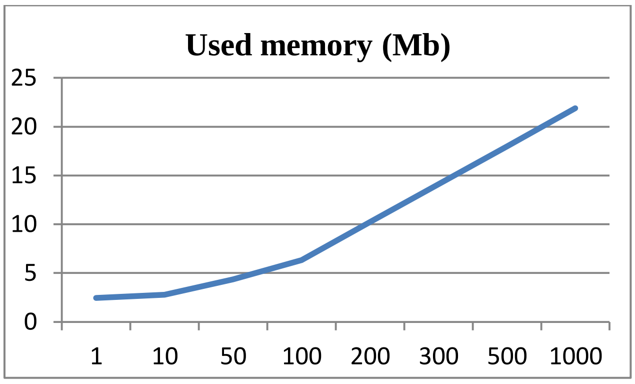

Figure 9: Memory use test result

The results got from the previous tests show that this framework can emulate a computer park with more than 2000 virtual machines in a real computer with average performance.

5.3 DDoS attack test

5.3.1 Test description

The aim of this test is to prove that this framework can emulate a DDoS attack using a huge number of bots and not to saturate the target machine bandwidth. So a mini Botnet was programmed (contains 2000 bots) in order to receive commands from a central server and execute them. This Botnet is holly based on LwCVM for both Server and bots that are connected to each other through a virtual router (NERouter). This router is connected to the Host machine network card using the connector RealNetworkConnector through a third virtual swith (NESwitch). Finally the target machine which is a real one was connected to the host machine using an Ethernet link. By that, all the machines

(Bots, Botnet server, target machine and the host machine) behave like if they are connected through the virtual router (see figure 9).

The network configuration is as follow:

- The Botnet server has the IP address 192.168.0.10 with the mask 255.255.255.0 and the default gateway 192.168.0.1
- The bots had a range of IP addresses from 10.1.0.10 to 10.1.7.224 with the mask 255.255.0.0 and the default gateway 10.1.0.1
- The host machine has the IP address 192.170.0.101 with the mask 255.255.255.0 and the default gateway 192.170.0.1
- The target machine has the IP address 192.170.0.100 with the mask 255.255.255.0 and the default gateway 192.170.0.1

The Botnet command was defined as follow:

ping IP_ADDRESS [-t]/[-n X]

Where:

- IP_ADDRESS: is the target host IP address.
- -n: option for sending the ping request X times
- -t: option for sending the ping request infinitely.

The following (figure 9) schema illustrates the test model:

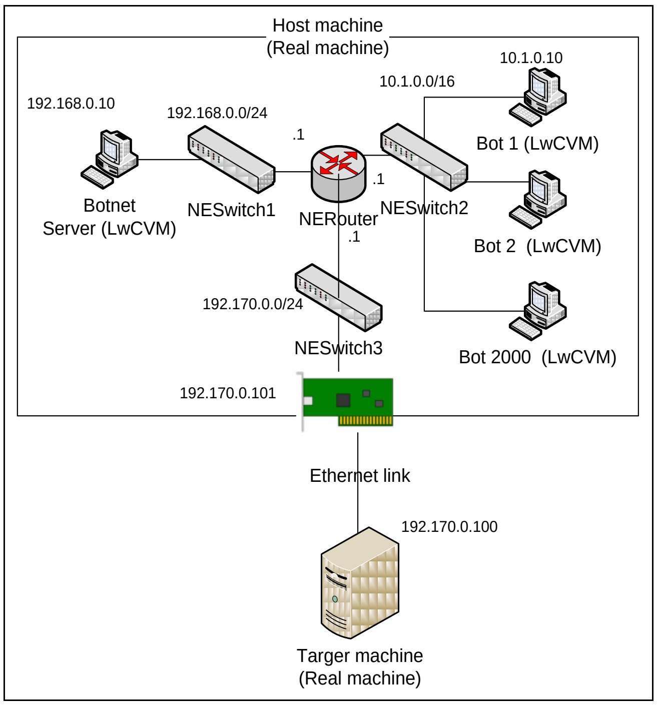

Figure 9: DDoS attack test model

5.3.2 Test result

The following chart (figure 10) generated using Wireshark illustrates the number of ICMP packets (1500 byte per packet) received by the target machine. It shows that the average packet rate is around 25000 packets per second. This rate is 25 times higher than Dynamips (GNS3).

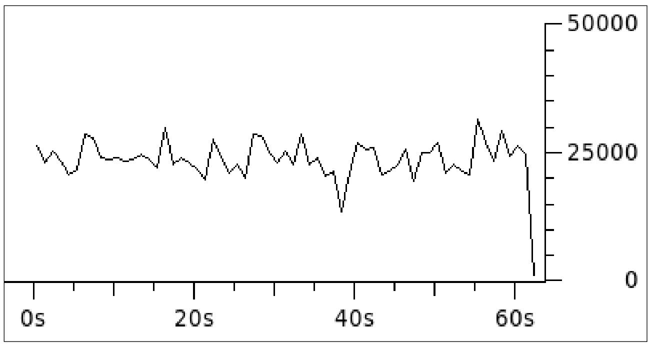

Figure 10: Packet rate test result

The following chart (figure 11) generated also using Wireshark illustrates connection bit rate received by the target machine in b/s. As noticed, the average packet rate is around 276 Mb/s.

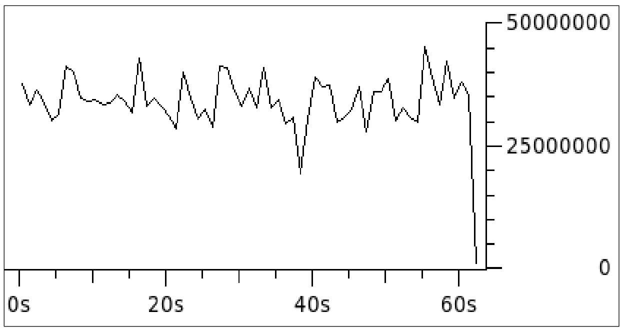

Figure 11: Bit rate test result

The results got from this test prove the ability of this framework to emulate a Botnet containing more than 2000 bots using a simple real machine.

During the test we noticed also that if the connector Real Network Connector is not used, the flow rate can reach in certain conditions 70000 packets/s which is equivalent a connection bit rate of 801 Mbps.

6. CONCLUSION AND FUTURE WORK

This paper presents a new scalable and distributed framework that meets the requirements defined previously by allowing the emulation of both networks and a huge number of virtual machines using normal computers. It can also be deployed over many machines to get a bigger computer park.

This framework was hosted in source forge under two projects Open source projects:

- Network emulator: http://sourceforge.net/projects/networkemulator/
- Lightweight C++ Virtual Machine: http://sourceforge.net/projects/lwcvm/

The reason behind this is to allow researchers and developers to use it, improve it and add new protocol stacks over the existing ones (like SIP, HTTP/S, telnet, SSH ….) because it is highly scalable.

Our future work will consist on developing an easy to use graphical user interface (GUI) in that allow users to drag and drop the available network equipments and connector with the possibility of customizing the behavior of the LwCVM and run them inside this network.

We will also develop IRC, peer2peer and SIP protocols over this framework in order to make a stress test for IPBX and also test some Botnet trace back and elimination algorithms.

7. REFERENCES

[1] Chao Li, Wei Jiang and Xin Zou. Botnet: Survey and Case Study.Innovative Computing, Information and Control (ICICIC), 2009

[2] Daniel Plohmann and Elmar Gerhards-Padilla.Case Study of the Miner Botnet. Cyber Conflict (CYCON), 2012

[3] Poongothai, M. Simulation and analysis of DDoS attacks. Emerging Trends in Science, Engineering and Technology (INCOSET), 2012

[4] Kihong Park and Heejo Lee. On the Effectiveness of Probabilistic Packet Marking for IP Traceback under Denial of Service Attack. INFOCOM, 2001

[5] Zahid M,Belmekki A and Mezrioui A. A new architecture for detecting DDoS/brute forcing attack and destroying the botnet behind. Multimedia Computing and Systems (ICMCS), 2012

[6] Overlay Simulation Framework official web site: http://www.oversim.org/

[7] Sandia National Laboratory official web site: https://share.sandia.gov/news/resources/news_releases/sandia-computer-scientists-successfully-boot-one-million-linux-kernels-as-virtual-machines

[8] Official GNS3 web site: http://www.gns3.net/documentation/gns3/introduction-to-gns3/

[9] Lightweigh IP official web site: http://savannah.nongnu.org/projects/lwip/

[10] Official Virtual PC simulator web site: http://sourceforge.net/projects/vpcs/

[11] POSIX Threads Programming official web site: https://computing.llnl.gov/tutorials/pthreads/