TD(08)006

The 11th COST 290 MC Meeting

Tampere University of Technology (TUT), Finland, May 27 – 28, 2008

WAN Emulator Software Tool

Andrei Bogdan Rus, Virgil Dobrota

Technical University of Cluj-Napoca, Communications Department
26-28 Baritiu Street, 400027 Cluj-Napoca, Romania
Tel/Fax: +40-264-597083
E-mails: {bogdan.rus, virgil.dobrota}@com.utcluj.ro

René Serral Gracià, Jordi Domingo-Pascual

Universitat Politecnica de Catalunya, Barcelona, Dept. d'Arquitectura de Computadors,
Jordi Girona 1-3, E-08034 Barcelona, Spain,
Tel: +34-93-4016981,
E-mails: {rserral, jordi.domingo@ac.upc.edu}

Abstract

Testing in real life conditions is not as easy as it first seems, because often it is costly and difficult to reproduce Internet behavior in controlled environment. WAN Emulator (WANE) proposed herein is a software tool that helps controlling the IP traffic parameters of a network, like delay, packet loss or packet duplication. The advantage of this software is that is using free tools implemented in Linux kernels but offers a friendly graphical interface, so that the researcher does not have to bother with time consuming operations like creating the `tc` tree, marking the specific traffic and filtering it.

Keywords

WAN emulator, IP traffic parameters, iptables, tc, One-Way Active Measurement Protocol

WAN Emulator Software Tool

Andrei Bogdan Rus, Virgil Dobrota, René Serral Gracià, Jordi Domingo-Pascual

Abstract

Testing in real life conditions is not as easy as it first seems, because often it is costly and difficult to reproduce Internet behavior in controlled environment. WAN Emulator (WANE) proposed herein is a software tool that helps controlling the IP traffic parameters of a network, like delay, packet loss or packet duplication. The advantage of this software is that is using free tools implemented in Linux kernels but offers a friendly graphical interface, so that the researcher does not have to bother with time consuming operations like creating the `tc` tree, marking the specific traffic and filtering it.

1. Introduction

Testing in real life conditions is not as easy as it first seems, because often it is costly and difficult to reproduce Internet behaviour in a controlled environment. The current tools available involve expensive hardware or proprietary solutions. However there is a software tool called NetEm *(Network Emulator)* which is an enhancement of the traffic control facilities provided by the Linux Kernel. The major advantage of this open-source software is related to its good performances. On the other hand, it is rather difficult to use NetEm, as it requests advanced knowledge in traffic control under Linux. The proposed tool, called WANE *(WAN Emulator)*, has a friendly user interface, while maintaining the good performance in evaluations.

The rest of the paper is structured as follows: the next section presents the software tool developed in this work, after this, we follow the discussion by explaining the basis of the core technologies used in this work, namely `iptables, tc` *(traffic control)* and NetEm Qdisc Design. Later we present the tests and results along with the limitations of the system. Finally we conclude and detail the further lines of research.

2. Software Tools: `iptables,tc` and NetEm

WANE is a software tool for Linux platforms and it is integrated into NetMeter, developed by Universitat Politecnica de Catalunya in Barcelona [1]. The software was designed to test the measuring tools already developed in NetMeter.

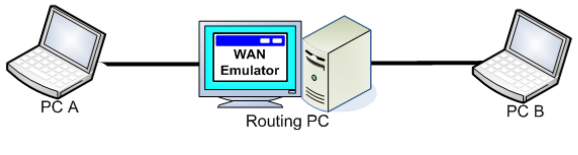

Figure 1. Testbed architecture

In Figure 1 we have illustrated the way that WANE is used in the testbed. The parameters that can be emulated with this tool are the following: delay, jitter, dropped packets and duplicate packets. Another facility that WANE offers is the ability to emulate several scenarios in the same time for different data differentiated by a series of parameters such as: source/destination address, mask and port; protocol (TCP, UDP or ICMP); input network interface of the router controlled by WANE.

The proposed emulator uses two tools implemented in Linux kernel: `iptables` and `tc`. In order to implement a specific scenario, the steps covered by the tool are listed below.

1. Marking the specific traffic flow for which the `tc` parameters are configured using `iptables.`
2. Building the `tc` tree composed by queuing disciplines and enabling the changes of IP parameters for the output data traffic. This step is implemented with the `tc` tool.
3. Filtering the flows with a specific mark previously set to the packet in the first step, and sending them to a specific branch from the `tc` tree built in step two. To implement it we used the filtering option of this tool.

`iptables` is the native packet filtering mechanism for the Linux 2.4 and above kernel series. We can use it to filter packets, implement network address translation and mangle packets. There are three default chains for filtering packets: INPUT, OUTPUT and FORWARD, and two default chains for network address translation: PREROUTING and POSTROUTING, and three tables: *filter*, *nat* and *mangle*.

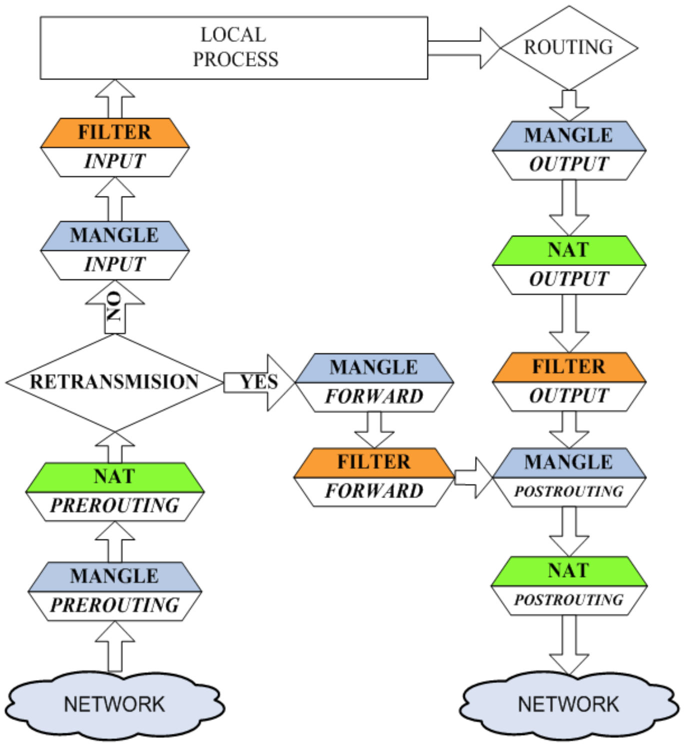

Figure 2. `iptables` architecture.

In Figure 2 we can see the chains through which the packets are going, when the `iptables` service is enabled in Linux kernels. In this paper we used the mangle table in order to mark packets from a specific data flow. The marking rules were set into the PREROUTING chain from the *mangle* table. We later used this mark in order to identify the flows and apply the desired changes in the network metrics.

There are three *targets* into the mangle table: TOS, TTL and MARK. TOS is used to modify the Type of Service field in the packet, TTL target can change the TTL (Time To Live) field from the header and MARK associates special mark values to the data flows. When using the MARK target, the value is not actually set into the packet but it is associated to the socket descriptor internally on the Linux kernel.

`tc` is an efficient and flexible software tool, implemented in Linux kernels, based on the concept of queue, i.e. a stack where the binary data is stored, before sending it to the network interface. The queues can have classes with different parameters and priorities. However the major issue is that it is complex to work with and requires some experience. One of the major goals of WANE tool is to make the user's job easier so the researchers will not have to bother with how the settings will be implemented, as long as they know exactly what parameters are needed and their exact values.

The position of Linux queuing disciplines into the TCP/IP stack is shown in to the Figure 3.

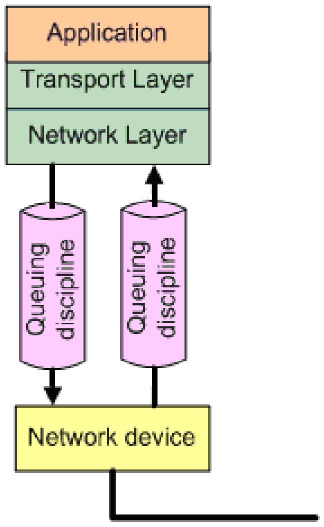

Figure 3. Layering of tc queuing disciplines.

A queuing discipline (qdisc) is a simple object with two key interfaces. The first one is putting the packets that need to be sent to the network, into the queue, and the other one, releases packets to the network device. `tc` supports two types of qdiscs: *classless* qdiscs (that can have multiple classes attached) and *classful* qdiscs (that have only one class attached). Each type of qdisc can be useful in different situations.

For instance, if we want to offer differentiated services to data flows, classful qdisc should be used. First we link a number of classes having different priorities or behaviors to the parent `qdisc`, and then we filter the packets by a set of parameters, and send them to one of these classes.

WAN Emulator uses a tc tree and builds it depending of the user needs, as in Figure 4. The root of the tree is a HTB qdisc used to limit the maximum transfer rate for the data flow. This option is used when the user wants to emulate a slower connection. By default, WANE will create a class with a maximum rate set to the link speed. Attached to the HTB class there is NetEm qdisc. This provides the functionality to emulate IP traffic parameters for testing several protocols. The current version emulates variable delay, loss and packet duplication [4].

A NetEm qdisc can emulate only one type of IP parameter, so in order to create a more complex behavior we have to cascade this type of qdiscs. If no qdisc is attached to the NetEm's class, we call this a *leaf class* because is the last one through which the traffic goes before being send to the network interface.

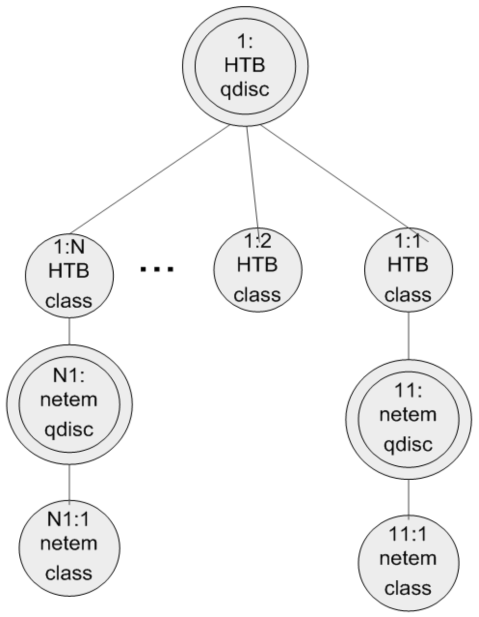

Figure 4. Traffic control tree build by WANE.

After creating the tree, the last step is to implement the filters used to discriminate the packets based on a set of parameters and send them to a specific branch of the tree that emulates a specific scenario. In this case, the packets are filtered according to their associated mark set by `iptables`.

NetEm is an enhancement of the `tc` tool and it can be used to emulate IP traffic parameters like: packet delays, dropped packets or duplicate packets. It consists of two parts, a small kernel module for a queuing discipline and a command line utility to configure it. The module has been integrated in Linux 2.6.8 kernels or later, and the command line utility is part of the traffic control package. Communication between the command line and the kernel module is done via the Netlink socket interface. Requests are encapsulated into a standard message format that is interpreted by the kernel.

3. OWAMP Testing Tool

OWAMP is one of the tools used to test the performance of WANE. It contains a command line client application and a policy daemon, using the protocol defined by RFC 4656 (A One-way Active Measurement Protocol) written by S. Shalunov, B. Teitelbaum, A. Karp, J. Boote, M. Zekauskas, in September 2006.

OWAMP measures unidirectional characteristics such as one-way delay, one-way loss and one-way duplications. High-precision measurement of these one-way IP performance metrics became possible with wider availability of good time synchronization sources (such as GPS and CDMA).

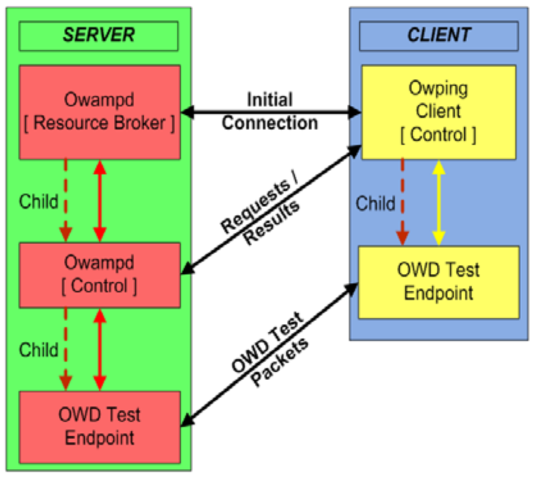

Figure 5. OWAMP Architecture

As it can be seen in Figure 5, OWAMP is a client-server application. The *owping* client contacts the *owampd* daemon on the peer host to request a specific test. This includes a series of traffic test parameters like: transmission rate, packet size in bytes and direction (from server to client or vice-versa).

owampd is responsible for accepting or denying the test requests. It was developed as a classic accept/fork daemon that listens for new network connections and also manages the resources for all child processes. Once a connection request is received, *owampd* forks a child process to handle it. As soon as a test session is accepted, the client and server both fork the OWD (One-Way Delay) Test Endpoint that actually sends and receives the packets used in the tests. These processes are implemented using identical code on both the server and the client.

When measuring the delay between a source and a destination, OWAMP tool writes into the data packets a timestamp and calculates the difference between the time when the packet was received and the time when the packet was sent. This is the reason why the NTP (Network Time Protocol) service must run on both source and destination computers, as their system clocks must be synchronized for meaningful measurement results. In this testbed we used Palisade GPS synchronization kit to synchronize the source and destination computers with the router PC which was the NTP server. The testbed including the synchronization kit can be seen in Figure 6:

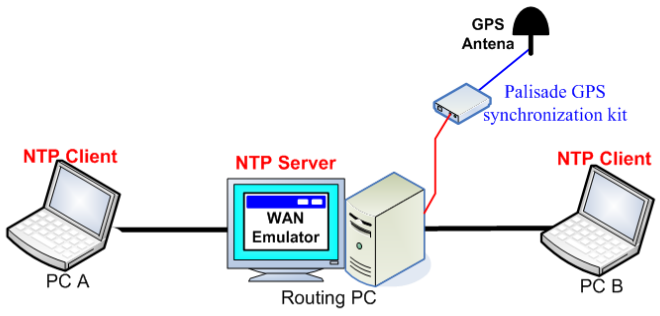

Figure 6. Testbed with Synchronization Kit

4. Tests And Results

We tested if the parameters (such as delay, percentages of dropped or duplicate packets and throughput) applied to WAN Emulator properly influenced the traffic flows.

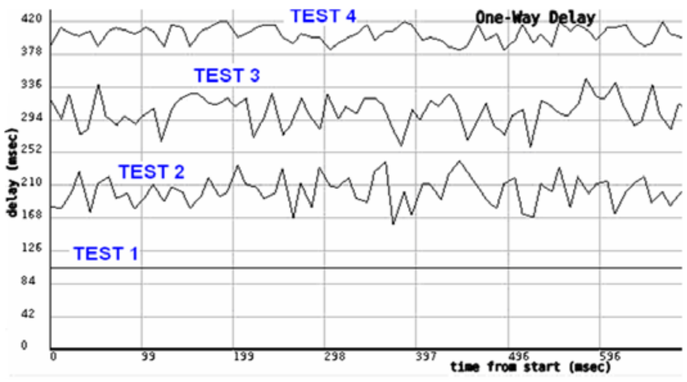

Figure 7. Experimental results: delay tests 1-4

First, we used the OWAMP tool in order to measured packet delay from a specific source to a destination. Within the first test in WANE we considered a fixed delay of 100 ms, whilst in the second test each packet was delayed randomly within a range of values 180 ms and 220 ms. Test 3 took into consideration the fact that the delay variation (jitter) is not purely random in real networks. So we specified a correlation between two consecutive values. This represented how much of the current delay applied to the packet depended on the previous one. This correlation is described analytically in the following equation:

$$Delay_n = Delay_{n-1} \cdot \frac{Corr}{100} \pm RandDelay \cdot \left(1 - \frac{Corr}{100}\right) \qquad (1)$$

where Delay$_n$ refers to the current packet and Delay$_{n-1}$ to the previous packet. *Corr* is the correlation and *RandDelay* is a random delay value. Within the third test the delay was in the range 280 ms up to 320 ms, with a correlation of 80%. Finally, the fourth test took the delay between 380 ms and 420 ms, with a normal distribution. Note that the *normal* distribution, also known as Gaussian, is a family of continuous probability distributions, defined by two parameters: the mean ("average") ,μ and the variance ("variability") σ^2, respectively. WANE implemented a mean of zero and a variance of one (the green curve in Figure 8).

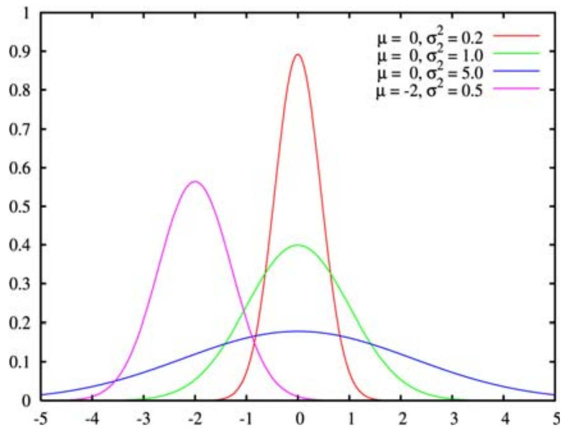

Figure 8. Normal (Gaussian) distribution

The density of probability function for the normal distribution is given by the equation:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

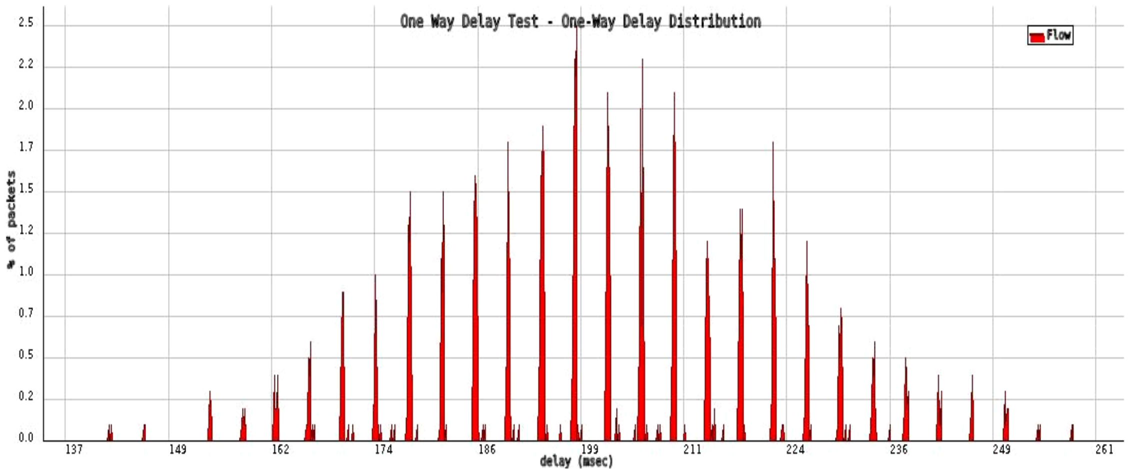

Figure 9. Normal distribution implemented by WANE

In Figure 8 we represented the percentages of packets delayed versus the delay (in milliseconds). This result proved that the emulator managed to impose a normal distribution.

Later on a Pareto distribution was analyzed too, in order to demonstrate that WANE is able to implement it. The graphical representation of the PDF (Probability Density Function) can be seen in Figure 10 and its definition is given by the formula:

$$f(x;k,x_m) = k\frac{x_m^k}{x^{k+1}} \quad for \quad x \geq x_m \tag{3}$$

for all x ≥ x_m, where x_m is the minimum possible value of x (necessarily positive) and k represents a positive parameter.

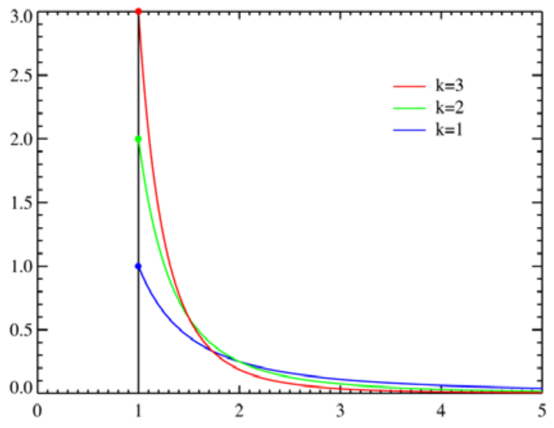

Figure 10. Pareto distribution

According to Figure 11, WANE can successfully emulate delays that have a Pareto distribution.

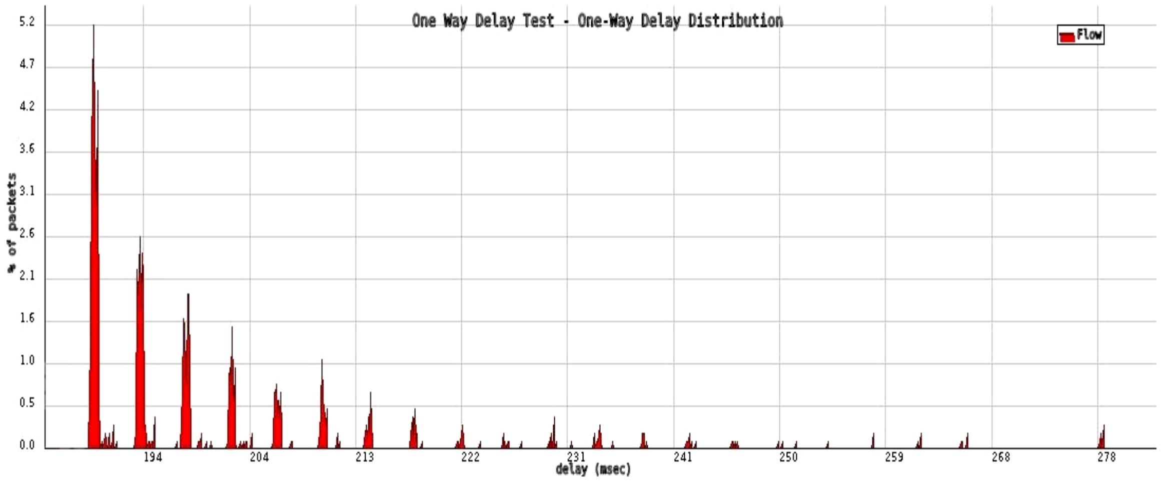

Figure 11. Pareto distribution

NetEm implements a specific discipline by taking a table to specify the distribution. The actual tables (normal, Pareto) are generated as part of the `tc` compilation and placed in `/usr/lib/tc` on the Linux machine. With additional effort it is even possible to make your own distribution based on experimental data and to store it in the same location as for the others [4].

The second group of experiments was devoted to check how accurate WANE can emulate packet dropping. Before doing a measurement we configured the emulator to discard a specific percentage of packets. For each set we performed three tests, taking the average as a final result. We also repeated the tests to see if the performance is the same if we vary the throughput. Observe from Table 1 that the difference between imposed values of dropped packets and the measured ones was less than 1%. This performance is good enough for emulating a certain connection as in a real network. The same procedure was applied for duplicate packets, the results being presented in the Table 2. Again the WANE emulation errors were less than 1%.

Value imposed by WANE [%]	Measured value at:			
	1kB/s [%]	10kB/s [%]	100kB/s [%]	1MB/s [%]
10	9.8	10.1	9.8	10.1
20	20.3	20.2	19.9	19.9
30	29.5	30.4	30.2	30.2
40	39.1	40.8	40.1	39.8
50	50.2	49.2	49.8	50.1
60	59.7	59.6	60.2	60.3
70	70.4	69.9	70.3	69.8
80	80.1	79.6	79.7	80.4
90	90.3	90.1	89.8	90

Table 1. Packets dropped

Value imposed by WANE [%]	Measured value at:			
	1kB/s [%]	10kB/s [%]	100kB/s [%]	1MB/s [%]
10	10.3	10.1	9.7	10.1
20	19.6	20	19.8	20.2
30	29.5	30.2	30.1	29.9
40	40.6	39.9	40.3	39.8
50	50.1	49.8	49.8	50.2
60	60.2	60.3	60.1	60.1
70	69.8	69.7	69.9	69.9
80	80.2	79.8	80.2	79.9
90	90.1	90.2	90.1	90.2

Table 2. Duplicate packets

Maximum throughput imposed by WANE	Average throughput measured	Error [%]
10 [kbps]	9.59 [kbps]	4.10
50 [kbps]	48.06 [kbps]	3.89
100 [kbps]	97.33 [kbps]	2.67
500 [kbps]	479.17 [kbps]	4.17
1 [Mbps]	0.95 [Mbps]	5.00
5 [Mbps]	4.79 [Mbps]	4.20
10 [Mbps]	9.58 [Mbps]	4.20
50 [Mbps]	47.89 [Mbps]	4.22
100 [Mbps]	95.95 [Mbps]	4.05

Table 3. Maximum throughput

Next we investigated the maximum transfer rate using a tool called NetPerf [5]. Thus we started from 10 kbps up to 100 Mbps. For each maximum rate imposed by WANE several tests were performed, the average throughput been presented in the Table 3. Observe that in this case the emulation error was less than 5%. There are several reasons for this. First is the fact that the measuring tool worked at the Application Layer and it did not count the headers added to the packets. Moreover TCP automatically adjusted the transmission window, usually underestimating the available throughput. The formula for the error value was the following:

$$er[\%] = 100 \cdot \frac{Val\_imposed - average}{Val\_imposed} \qquad (4)$$

Regarding the limitations of WAN Emulator tool, these are due to the disadvantages inherited from NetEm qdisc. So, NetEm does its best to a single flow. However, real world networks are quite complex and the emulation inevitably breaks down in some circumstances. Linux version used was not a real-time operating system and this generates some constraints on the performance of a real-time simulator such as NetEm. Kernel timers are limited by the system time tick rate of 1000 Hz (1 ms) on Linux 2.6. The Linux 2.4 kernel uses a slower 100 Hz clock (10 ms). Therefore NetEm can not be used to emulate relatively short delay networks of less than 1 ms [6].

6. Conclusions

NetEm (Network Emulator) proved to be a useful tool for testing protocol behaviour. It provided the necessary statistical options to emulate real world network response. The author of NetEm developed this tool, in order to validate BIC TCP and TCP Vegas protocols for the Linux 2.6 kernel. But since then, many other developers used NetEm to test protocols and applications. The proposed tool WANE (WAN Emulator) made the job easier for the researchers because they will not have to bother with creating the tc tree, marking the specific traffic and filtering it. All these operations are made by WANE in a decent time with the same accuracy as NetEm. As future work, we want to improve WANE in order to support IPv6 traffic. Additionally we envisage an extra feature of the emulator to be able to corrupt packets and to test how this could influence the quality of the tested flow

References

[1] ***, http://www.ccaba.upc.edu/netmeter
[2] http://www.ks.uni-freiburg.de/download/inetworkSS04/practical/iptables-intro-short.pdf
[3] MA.Brown, Traffic Control HOWTO, http://linux-ip.net/articles/Traffic-Control-HOWTO/classful-qdiscs.html, 2006
[4] ***,http://linux-net.osdl.org/index.php/Netem
[5] ***,http://www.netperf.org/netperf/training/Netperf.html
[6] S.Hemminger, *Network Emulation with NetEm*, 2005
[7] J.Boxman, *A Practical Guide to Linux Traffic Control*, Chapter 6.3 Using the Netfilter MARK Target, http://trekweb.com/~jasonb/articles/traffic_shaping/ classlows.html
[8] B.Hubert, Linux Advanced Routing & Traffic Control HOWTO, http://lartc.org/howto/