WAN EMULATOR

Andrei Bogdan Rus [1], *René Serral Gracià* [2], *Jordi Domingo-Pascual* [2], *Virgil Dobrota* [1]

[1] Technical University of Cluj-Napoca, Communications Department, 26-28 George Baritiu Street, 400027 Cluj-Napoca, Romania, Tel:+40-264-401226, E-mails: {bogdan.rus, virgil.dobrota}@com.utcluj.ro}

[2] Universitat Politecnica de Catalunya, Barcelona, Dept. d'Arquitectura de Computadors, Campus Nord. Modul D6. 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 behaviour 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.

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.

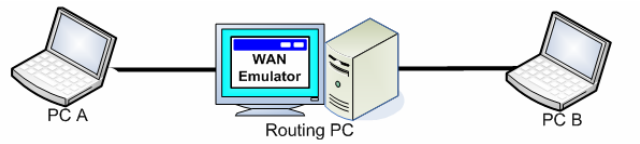

Figure 1. Testbed architecture

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.

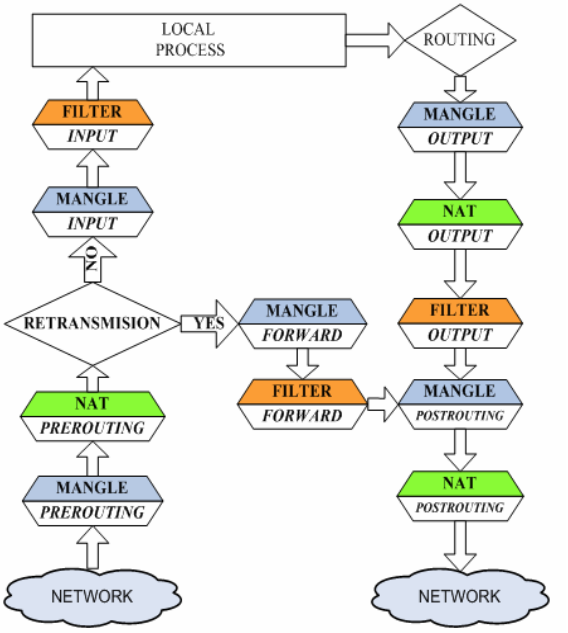

Figure 2. *iptables* architecture

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*.

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. We later used this mark in order to identify the flows and apply the desired changes in the network metrics.

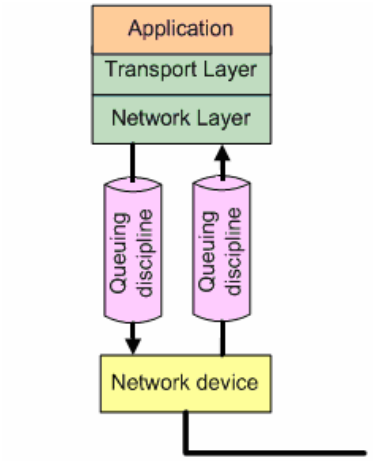

Figure 3. Layering of *tc* queuing disciplines

tc is a software tool, implemented in Linux kernel, 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 it requires some experience. One of the most important goals of WANE tool is to make the user's job easier.

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 set a maximum transfer rate for the data flow. These parameters are used when WAN Emulator builds the class, attached to the `root qdisc`. If the user does not want to specify any of the above parameters, WANE will create a class with a maximum rate set to the link speed, so that these settings will not affect in any way the traffic that is forwarded to this class. A NetEm qdisc was attached to the HTB class, as in Figure 4. This provides the functionality to emulate IP traffic parameters for testing several protocols. The current version emulates variable delay, loss and packet duplication [4].

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

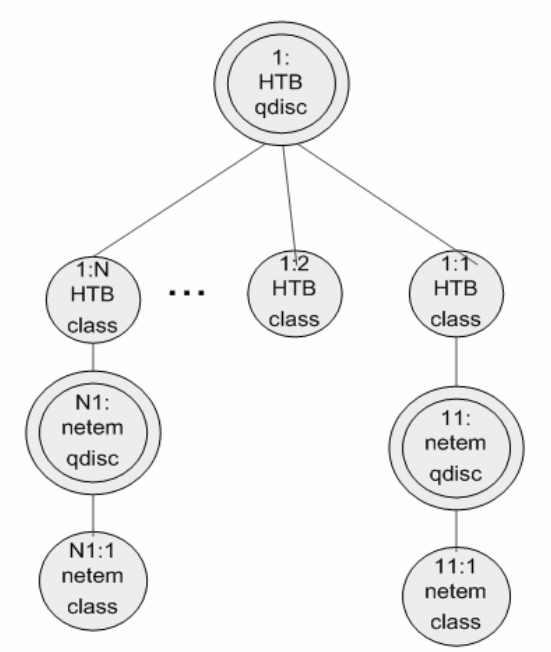

Figure 4. Traffic control tree build by WANE

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 kernel module has been integrated in Linux 2.6.8 kernels or later.

3. 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.

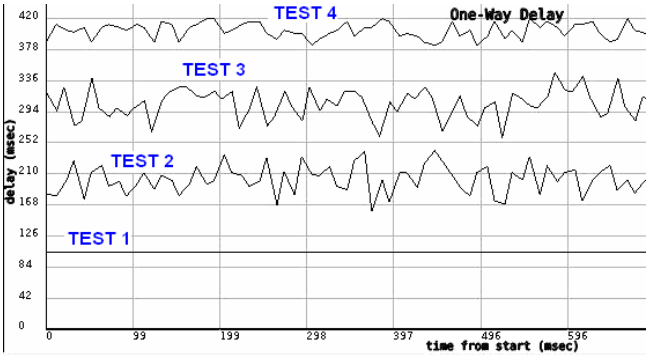

Figure 5. Experimental results: delay tests 1-4

First, we used a tool from NetMeter that 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:

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

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 (Gaussian) distribution.

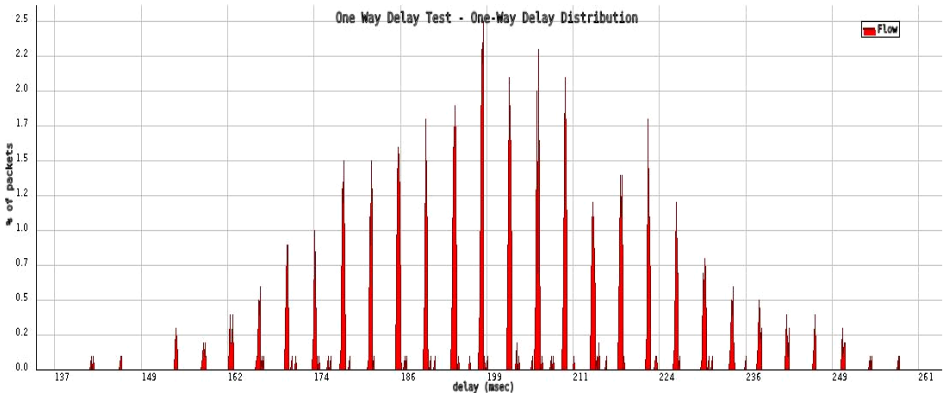

Figure 6. Normal distribution

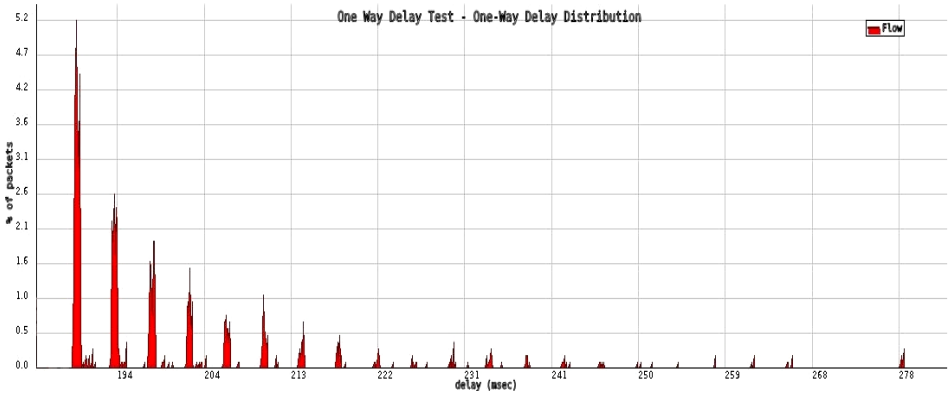

Figure 7. Pareto distribution

In Figure 6 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, demonstrating that WANE is able to implement it (see Figure 7).

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.

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%. 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.

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

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:

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

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].

4. 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] ***,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/