U2-Route: A Hardware Tool to test Internet Quality of Service proposals

J. Padilla
Universidad Pontificia Bolivariana
Km 7 Via Piedecuesta
+57-76796220
jhon.padilla@upb.edu.co

L. Santamaría, C. Acevedo
Universidad Pontificia Bolivariana
Km 7 Via Piedecuesta
+57-76796220code
luis.santamaria@upb.edu.co

L. Becerra
Universidad Católica de Pereira
Kra 21 #49-95 Av. de las Américas
+57-63127722
line.becerra@ucpr.edu.co

ABSTRACT

In this paper, we describe a new hardware tool to test new Quality of Service proposals. Thus, we want to reduce the digital divide between Latin America development countries. The problem is that in our region it is difficult to build hardware prototypes for new networking protocols and methods. Our tool, named U2-Route, is based on a NetFPGA board, an open source hardware system developed in Stanford University. U2-Route has a routing core and a Web Interface that allows research groups belonging to Universities to test new developments. As an example, we show the methodology to design and to implement a Classifier and the results obtained.[1]

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking – *Routers.*

General Terms

Algorithms, Measurement, Performance, Design, Experimentation, Verification, Internet, Routing.

Keywords

Quality of Service, Hardware development Tools, FPGA, Router Architecture, Remote Lab.

1. INTRODUCTION

Quality of Service area has received many contributions in last years. However, many proposals were evaluated only by means of simulations or mathematical models. They suffer of hardware prototypes lack. This problem in development countries is bigger than other countries in the first world. To improve this aspect of the digital divide in Latin America, we have been working on a remote lab that allows several universities connected to RENATA network in Colombia and CLARA network in Latin America, to make hardware testing for several proposals in Internet Quality of Service area. Such proposals can be loaded in our hardware testbed, then it can be proved and results could be read in several text files. Results cover aspects such as packet delay, packet losses, bandwidth, etc.

As an example of how to use our tool to test Quality of Service methods, we have developed a Classifier module. This is a partial result in our research, because then we want to develop a Differentiated Services Router.

An important aspect of our Tool is that it is based on a NetFPGA board [1]. This board was designed in Stanford University and its main objective is to bring an open source hardware Tool. Our work took an IPv4 reference Router allocated at NetFPGA project web page and then, we modified the source code to insert our QoS support modules. The transfer of technology really was a hard aspect due to the complexity of tools required to implement new modifications to the NetFPGA hardware. However, at this point, we are capable of design and teach to our students about the process and now, we are using this tool to teach about Router Architecture, Quality of Service and other related topics. Then, we want to offer our tool to the Latin America network researcher community. At this moment, the Web Interface is working under restricted permissions because we are performing connectivity tests, so now is not possible to connect new users to our platform. However, the idea is that in some months, we will to put our web interface on line with the world by means of RENATA and CLARA networks for free access to research groups belonging to universities.

This paper has the following parts. Section 2 describes the U2-Route project parts. Section 3 detailed the Web Interface. Section 4 describes the NetFPGA board. Section 5 describes the IPv4 reference model. Section 6 detailed the main modules required to support Quality of Service in a Differentiated Services manner. Section 7 describes our Classifier module design with FPGAs. Section 8 shows some results for our Classifier marker. Finally, in section 9, we provide our conclusions.

2. RELATED WORK

There are several possibilities to build Academic Routers. First, there are solutions by software. Many Software Routers are based on Linux. There are two important software Routers that are common tools: XORP project and CLICK project. XORP project [10], is a flexible platform but it suffers a complex human interface and documentation lack. Other important software tool is the CLICK project [11]. Such project is a very common tool.

LANC 2011, October 12-13, 2011 Quito, Ecuador

However, both approaches have a great disadvantage: each packet should be processed by several software layers. Then, software routers are slower than hardware routers.

Second, Hardware solutions for routers have two approaches: Network Processor approaches and FPGA processing approaches. The first approach is based on Network Processors; in consequence, such approach requires software programming. Although such processors are designed for packet processing work, they execute all the steps instruction by instruction. Also, such tools are expensive and they are difficult to acquire in Development countries. On the other side, the FPGA based solutions for router building are flexible and they work in parallel for packet processing. Also, there are FPGA based solutions, such as NetFPGA, that are cheap due to sponsorship from manufacturers.

3. U2-ROUTE PROJECT PARTS

This project developed a platform to evaluate cutting-edge networking technology proposals in telecommunications research. Such new approaches are fundamental for the next generation networks development.

U2-Route system is primarily composed of a networking hardware core, where all the hardware designs and testing is made, and also, U2-Route is composed by a web interface to interconnect the hardware core to Internet users. The hardware core is based on a platform called NetFPGA [1], that allows to design and test different modules in an Internet Router. The web interface interconnect the NetFPGA platform with the users connected to RENATA network, which allows to different research groups in Colombia and Latin America to test their own developments in this area and obtain results based in a real device.

4. THE WEB INTERFACE

The web interface, which appears in Figure 1, brings the connection between the Hardware tool and the users in the outside world. This connection works through the RENATA Network, where research groups around the country can access and use it. The main idea is that universities can develop their own proposals for QoS and other mechanisms; then, they send their own Hardware code described with an HDL (Hardware Description Language) through the Web interface in order to be compiled and tested. Once the compilation is made and tested, the web interface sends back the results to the research group. Then, it is possible to obtain results for several parameters and several types of traffic. Inside the website there is the Reference Router available to download (this model is explained later in detail), in consequence, all the research groups don't have to start building their own Router from scratch. They would just have to add their new module and connect it to the modular design in the Router.

Once the operation of a remote laboratory was analyzed, we decided to subdivide the web interface software into four modules; this division was made in order to make possible the approach of each of the factors that make up the overall software structure.

- User Management Module: We design and implement an application for user administration and tests management in a web environment for the remote laboratory of packet processing.
- Tests Generation Module: We design and implement a web application that allows capturing, storage and sending parameters to perform QoS experiments in the remote laboratory.
- Report Generator Module: It is a web-based report generator to present the data provided by the remote laboratory, presenting the results in a graphical mode.
- Traffic Generator Module: We build a traffic generator that generate traffic of different kinds and inject it to the router core.

5. NETFPGA BOARD

The testing core is based on the NetFPGA card [1]. The NetFPGA is a card designed at Stanford University. It consists basically of an FPGA chip named Virtex-II Pro 50, memories and four Gigabit Ethernet ports. The hardware can process packets at the maximum transfer rate of the links, reading and processing the packet headers for routing. The card is inserted into a computer through a PCI port (See Figure 2). The operating system used on the computer is the Linux Centos 5.4.

Figure 1. U2-Route Web Interface.

The card can be used to build routers for teaching or to conduct research. The hardware modules for FPGAs (gateware) are builded with an open-source philosophy; then, software and hardware are freely downloaded at NetFPGA project page [1].

All projects developed by different research groups in the world are shared in the community NetFPGA [2], which involved prestigious universities around the world. In that community, there is a repository with all code for the hardware of the various investigations, which is available to other universities to access and make improvements on them. Among the most outstanding designs are the named "reference projects": an IPv4 router, a switch and a network card (NIC-Network Interface Card for its acronym in English). To be able to work on these other projects, these should be written in a modular fashion to make it easy to build a new module within the design taking care of the standard interconnectivity between the modules. Figure 3 shows, as you can see, the modules of a given project within the NetFPGA.

Modules in the Pipeline (line processing) of the IP packets are interconnected by two buses: the Register bus and the packet bus

[3]. The packet bus sends data packets between different modules in which processing are performed. Data are sent as a series of words of 64 bits containing a header and a payload of the packet which is processed as a stream of data within each module. Moreover, the register bus interchange registers, enable registers, counters and tables from hardware to software. Software also allows that hardware change these elements. Register bus reads and writes through a ring of modules as shown in Figure 3. Any stage of the chain of modules is authorized to issue records requests for access module so module i can get information from module i + k if required.

Reference projects can be reused and shared with the academic community. Then, regarding standardization, a project must have an organization within the Linux directory, as shown in Figure 4.

All documentation for each project should be allocated in the doc directory under the project. In the folder "include", Project is the XML file (project.xml) that names and describes the project; it contains all shared verilog modules used by the project and defines the location of all records in project module.

Figure 2. NetFPGA PC

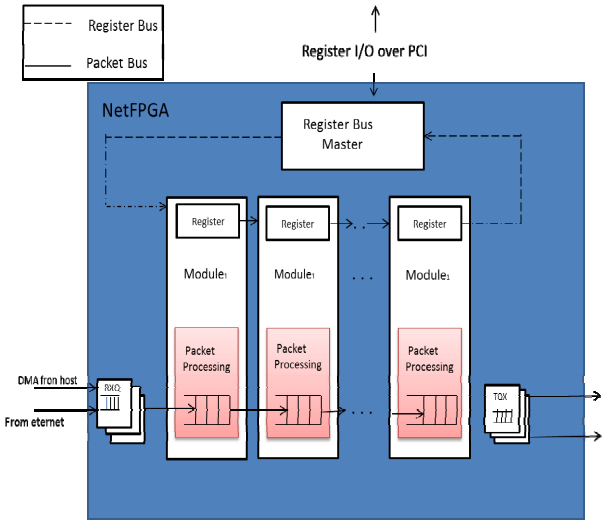

Figure 3. Pipeline. Register Bus and Packet Bus [3]

This directory also contains all the files of XML modules that are needed for a specific project. All modules of a specific project are in the "src" folder. The NetFPGA ensure that the modules within the project's "src" folder overwriting any module having the same name that is included in the shared modules.

Besides, "Lib libraries folder" is used by NetFPGA scripts to store the log files generator packages for both C and Perl. This directory and "log" files are automatically generated when running the simulation and synthesis. "Synth" folder is where you will find all files with Xilinx extension ("Xic"). These files are used to create cores (reusable modules) from your Xilinx Core Generator program. Also, within this folder, is where the project is synthesized by "make" command in the folder. Once generated the file with extension ".Bit", you can perform regression tests on this file. Such tests have as main objective to verify the correct operation of the new developed system. The regression tests are within the "regress" folder and are also used to specify the characteristics of a project and verify its functionality.

Verif folder (from "verification" word) is the directory where the simulation tests used to verify the performance of gateware. The evidence is recorded in separate directories within this folder.

netfpga	[base directory]
projects	[project directory]
<project_name>	[contributed project]
doc	[documentation]
include	[project.xml, project specific module XML]
lib	[perl and C headers]
src	[non library verilog]
synth	[all .xco files]
regress	[regression tests]
verif	[simulation tests]
sw	[project software]

Figure 4. Directory Tree for NetFPGA projects

Naming schemes for the tests are named "minor major test", where major and minor labels can be named like the project designer. The simulation script uses the major and minor labels in order to determine simulation run order. Each test within the regression test has a file called "make_packets.pl", which uses Perl libraries included in the NetFPGA package to create packages for testing data and read / write registers for simulation. Perl scripts are used for two purposes:

- Generate text files used as input to the Verilog simulation. These files describe packets that must arrive at the system and also when they should arrive. It also describes PCI bus events (eg, read / write).

- The other function is to compare the packets transmitted from a text file containing a list of packages expected.

6. IPV4 REFERENCE INTERNET ROUTER

The IPV4 reference router has a hardware design that consists of a modular pipeline with multiple components that can be reused by new projects. This gives a special advantage for researchers that want to add different functionality and want to prove their own features rather than start from scratch, building a router from the beginning. The modular design allows the components to be easily shared, thus facilitating the development of projects [4].

The reference router is a 5-stage pipeline (see Figure 5). The first stage of the pipeline has the Rx queue. These queues receive

packets from the Ethernet ports and from PCI interface through the DMA (Direct Memory Access) procedure. The ports are connected in a container called the User Data Path, which contains the processing stages. In the User Data Path, the first module which a packet passes through is the Input Arbiter. The Input Arbiter decides which Rx queue bring its packet, so it gets the package from this queue and passes it to the next module in the Pipeline, which is the Output Port Lookup module. The Output Port Lookup module is responsible to decide which port forward the current packet. After taking this decision, the packet is delivered to the Output Queues module, which stores the packet in the output queue corresponding to the output port until the Tx queue is ready to accept the packet for transmission [1].

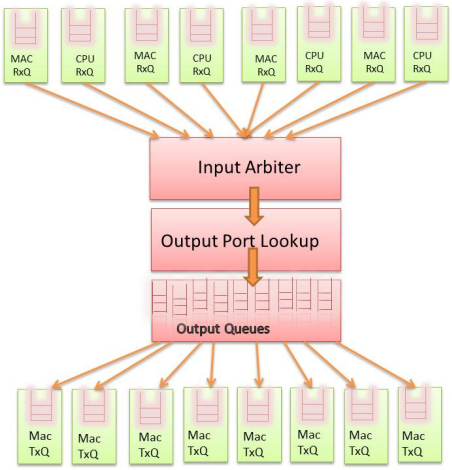

Figure 5. User Data Path [4].

7. MODULES REQUIRED TO SUPPORT QOS

Quality of Service (QoS) concept is being applied in recent years to obtain better management of data networks. With the arrival of technologies that allow several services over IP networks, QoS capabilities are required in a network to support real time services such as VoIP, IPTV, video streaming, etc, at the same time with the traditional elastic services such as Web, e-mail, FTP, etc. [5]

The first step in learning QoS support in routers or switches is to understand the modules that are necessary to support such capabilities. Those modules are [6]:

Packet Classification: Normally, a packet belongs to a flow or stream of packets. Such flow is composed for several packets sent from one source application to one destination application. Then, in a router that brings a special treatment to each flow, as is required in QoS routers, it is necessary to separate arriving packets in such flows to deliver them to the next module: the scheduler.

Packet Scheduler: This module shares the output channel among several flows. There are different approaches to complain this function. Some common schedulers are WFQ (Weighted Fair Queueing) and SP (Strict Priority Queueing).

Packet marker: In routers that receives directly the packets from end users, it is necessary to mark packets that are sent close to the maximum allowed rate. These packets should be marked because in other parts within the network these packets could be dropped if there is a congestion situation.

Packet dropper: One user could send data at bit rates higher than the maximum allowed by the network; usually this amount is described in the SLA (Service Level Agreement). Then, this module drops packets that are out of the maximum rate.

8. CLASSIFIER DESIGN

In order to give a first step in providing QoS we have designed a Classifier that classifies the traffic according to different parameters such as protocol, source port and destination port. Our module Classifier is interconnected inside the User Data Path between the Input Arbiter module and the Output Port Lookup module as you can see in Figure 6.

The Classifier basically takes a packet from the Input Arbiter, then it checks the parameters and finally, it modifies the DSCP (Differentiated Services Code Point) field inside the TOS (Type of Service) Field of the IPv4 header. We are based on Table 1 to mark the packets according to the type of traffic. This table shows the standard DSCP values for several types of Services as appear in [7]. This step is indispensable in order to provide QoS in a Differentiated Services Router. Thus, we will add other new modules that take actions over different traffic classes in the network, such modules are traffic shapers, schedulers, etc.

In order to understand the Classifier operation, it is necessary to describe how the interface between modules operates. Modules in the pipeline require interchanging the packets between them. To do this, a packet header should be divided in 64 bit parts. Each part should be accomplished with a signaling word (called Module Header) that helps the next model to understand some parameters required for its function. This information interchange requires a star-finish protocol to communicate the modules.

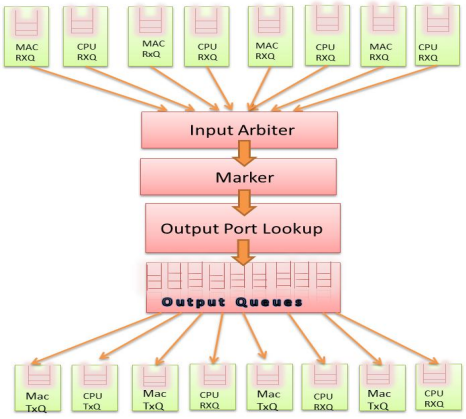

Figure 6. User Data Path with Marker

In figure 7 you can see the Finite State Machine (FSM) that describes the digital circuits used inside the Classifier module. Basically, it starts receiving the packet, starting by the Module Headers and the Ethernet Header. Then, the IPv4 header is received; when the Protocol field in the IP header is detected, the state changes to the "Store_start" state. At this point, the Classifier stores the entire IP header until the source port and destination port have been read. Then, it goes to the next stage Calc_TOS where it compares the source port and destination with the entries in a table where all rules are saved. When a rule is

found, the DSCP camp is changed to the new corresponding value.

Finally the module starts sending again the IP header and the rest of the payload to the next module.

9. RESULTS

In order to test our Classifier, we perform two types of proofs. First, we develop several simulations in the Modelsim simulator for FPGAs systems. Then, we downloaded our HDL design to the NetFPGA card and we generate packets to a Gigabit Ethernet input port in the NetFPGA. Then we capture the packets in the Gigabit Ethernet output port. Such captures were obtained with the WireShark Protocol Analyzer.

Table I. Standard Values for DSCP.

Service Class	DSCP	Value	Traffic Conditioning	PHB	Scheduler
Network Control	CS6	110000		RFC 2474	WRR/WFQ
Telephony	EF	101110	Policy control (one rate, burst size)	RFC 3246	SP
Signaling	CS5	101000	Policy control (one rate, burst size)	RFC 2474	WRR/WFQ
Multimedia Conference	AF41 AF42 AF43	100010 100100 100110	Low rate, three color marker (as in RFC 2698)	RFC 2597	WRR/WFQ
Interactive real time	CS4	100000	Policy control (one rate, burst size)	RFC 2474	WRR/WFQ
Multimedia Streaming	AF31 AF32 AF33	011010 011100 011110	Low rate, three color marker (as in RFC 2698)	RFC 2597	WRR/WFQ
Video Broadcasting	CS3	011000	Policy control (one rate, burst size)	RFC 2474	WRR/WFQ
Low delay data	AF21 AF22 AF23	010010 010100 010110	one rate, three color marker (as in RFC 2697)	RFC 2597	WRR/WFQ
OAM	CS2	010000	Policy control (one rate, burst size)	RFC 2474	WRR/WFQ
High rate transmission	AF11 AF12 AF13	001010 001100 001110	Low rate, three color marker (as in RFC 2698)	RFC 2597	WRR/WFQ
Standard	DF	000000	N/A	RFC 2474	WRR/WFQ
Low Priority Data	CS1	000000	N/A	RFC 3662	WRR/WFQ

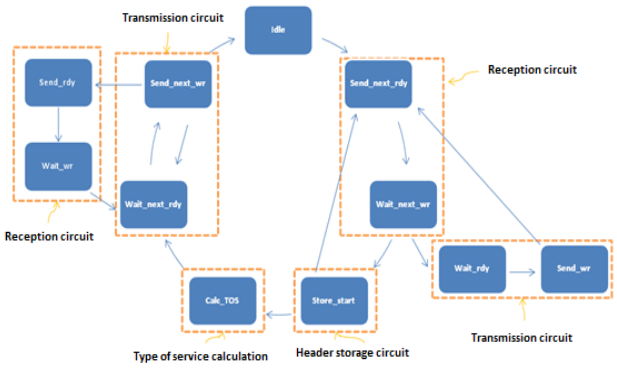

Figure 7. Classifier Logic

The simulations of the circuit are shown in figures 8 and 9. Now, we will describe the obtained results in this project phase.

The first part of our Classifier module is the reception step where we start getting the package from the Input Arbiter (see Figure 8). The signal *in_crtl* tells when a module header is coming; when *in_crtl* is high, it indicates that incoming data is part of the packet header. On the other hand, when data is a part of the IP packet load, *in_ctrl* is set to zero. Other signals are the *in_wr* and *in_rdy* signals, which indicate when the previous module is ready to send, and when our module is ready to receive respectively.

The *in_data* signal contains the data that is coming from the previous module; besides, the *out_data* shows what is coming out from our module. You can see that our module starts sending to the next module all the data until it gets the third word from the *in_data*. This is because in this word is allocated the DSCP field in the last bits. In this moment, the *out_wr* is established to zero in order to storage the packet. This signal remains in such state until the Classifier extract the information of the protocol contained inside the IP header, and extracts the source port and destination port stored in the transport layer header. For this reason, we have to check the IP header length to see if the IP header comes with options.

After the Classifier obtains the protocol identification, the source port and the destination port (all they conform a "tuple"), it compares these values against a table allocated in a registers set named *quinta* and *quintb,* where all rules are saved. These rules are defined by the network administrator; hopefully, according to the recommendation of Table 1.

When the Tuple matches with a rule in a table, the Classifier take the new DSCP value and write it into the IP packet header, which has previously stored. Then, our module starts again the transmission step, sending to the next module the rest of the IP header and the rest of the payload.

In order to see the actual result, we use the Wireshark Protocol Analyzer [9], which uses the "pcap" file format [8] for storing packet information

In figure 10 it is shown how a generated packet (left side of the Figure) was marked (right side of the Figure).

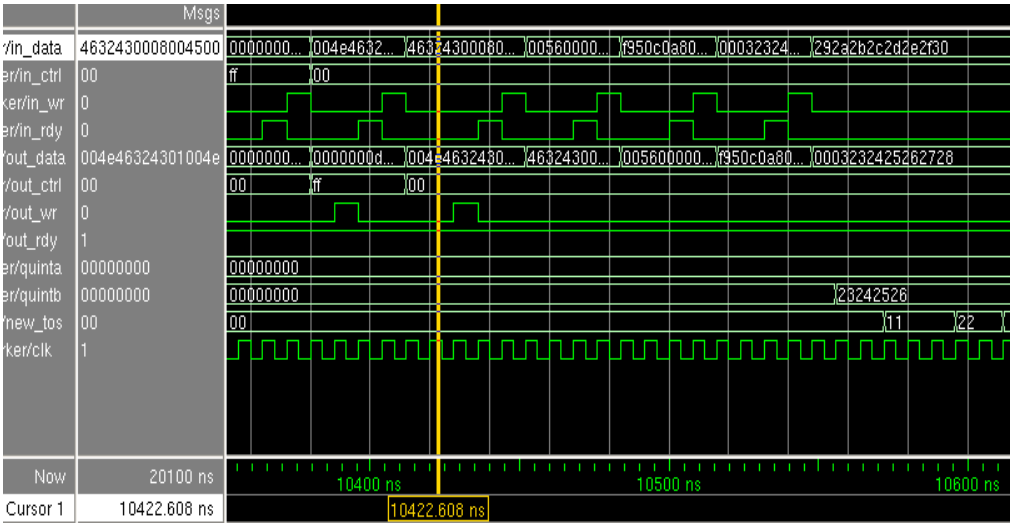

Figure 8. Simulation part 1

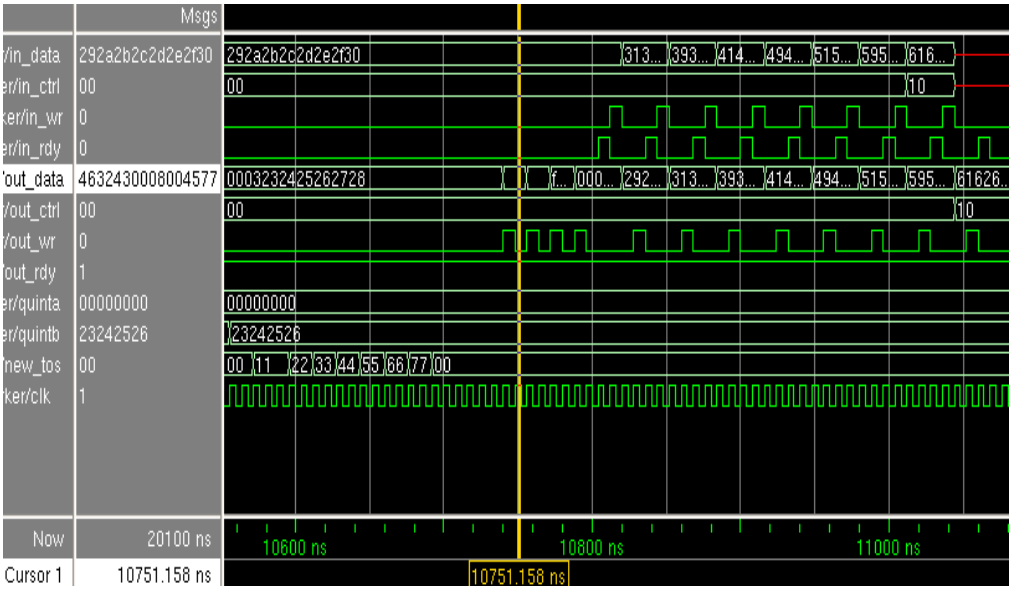

Figure 9. Simulation part 2

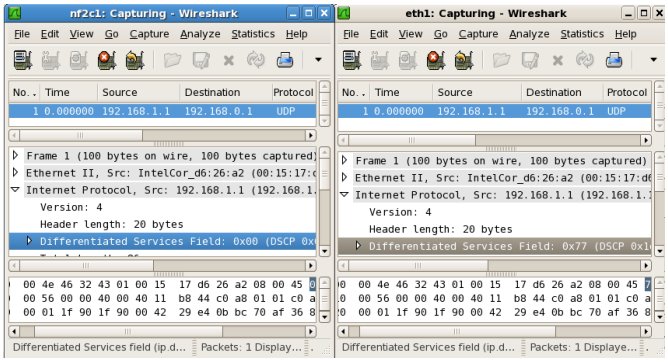

Figure 10. Packet passing through the Classifier

In this case, you can see that a packet is being sent by the interface n2fc1 and it has a value in the DSCP of 0x00. Then the packet is received by the interface eht1 and the DSCP is already changed to 0x77. We assign this experimental value in order to check that the marking was being realized correctly.

10. CONCLUSIONS

This paper presents a platform called U2-Route, consisting of a core test and a remote interface. This system has been developed with the support of Colciencias (Colombia Science and Technology Department) and RENATA (Colombia's High Speed National Network). By means of this project, we want to reduce the digital divide between Latin America and the developed Countries in the Data Communications Research area.

The core test of our project is based on a hardware tool that allows developing and testing different features and new protocols over IP devices. This platform, known as NetFPGA, is Open Source and it uses FPGAs to build the hardware for packet processing. The advantage of this tool is that you can build a hardware prototype of network devices that perform different types of functions offered by the research groups. Hardware as a tool, allows you to process packets at full line-rate as these do not

require going through the different layers of communication software of a computer to perform the desired functions.

As an example for the implementing quality of service, it is presented the case of a Classifier in order to classify the traffic. This feature was added to a pre-builted IPv4 Router project which is available on the NetFPGA community page. This document explains the process required to add a new module to reference projects available for NetFPGA. Finally, we showed some results of tests with the Classifier module that was added.

11. ACKNOWLEDGEMENTS

This Project is supported by Departamento Administrativo de Ciencia y Tecnología de Colombia- Colciencias, and by RENATA-Red Nacional de Alta Velocidad de Colombia, through the 487 grant (2009). This Project was performed by the Universidad Pontificia Bolivariana and the Universidad Católica de Pereira.

12. REFERENCES

[1] NetFPGA Development Team. NetFPGA website. http://www.netfpga.org/

[2] Contribuyendo con aplicaciones: http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/ContributingApplications

[3] G. Adam Covington, Glen Gibb, Jad Naous, John W. Lockwood, Nick McKeown Encouraging Reusable Network Hardware Design

[4] Glen Gibb, *Member, IEEE*, John W. Lockwood, *Member, IEEE*, Jad Naous, Paul Hartke, and Nick McKeown, *Fellow, IEEE,* NetFPGA—An Open Platform for Teaching How to Build Gigabit-Rate Network Switches and Routers, IEEE TRANSACTIONS ON EDUCATION, VOL. 51, NO. 3, AUGUST 2008

[5] J. Padilla, Y. Contreras, L. Santamaria; A Low Cost QoS Laboratory for Development Countries; IEEE LATINCOM conference; Bogota, COL; September 14-17, 2010

[6] D. Comer. "Network Systems Design using Processor Networks". Pearson Education. 2006.

[7] RFC 4594, Configuration Guidelines for DiffServ Service Classes, August 2006.

[8] libpcap packet capture library. http://www.tcpdump.org

[9] Wireshark. www.wireshark.com

[10] XORP project page: www.xorp.org

[11] The click modular router page: http://read.cs.ucla.edu/click/click