Ethernet Switch/terminal Simulators for Novices to Learn Computer Networks

Yasusi Kanada
Kogakuin University, Tokyo, Japan
yasusi@kanadas.com

***Abstract* – Many commercial and open-source network simulators are available; however, most of them are not suited for novices in college and school education or learning. Moreover, although Ethernet has recently become much more important, simulators that can be used for this purpose is rare because most simulators suited for education and learning are designed for IP communication. The author developed a simple CLI-based Ethernet simulator that can display contents of Ethernet packets to send or to receive in terminals and contents of MAC address tables in switches in "real time", and used the simulator in a university class for learning computer networks. The simulator, which is in public domain, is written in Python and, thus, runs on Windows, Macintosh, Linux, and other operating systems. The use of this simulator was evaluated based on a report assignment. The evaluation result shows that the average score of the reports written by using the simulator was much better, although it is not statistically significant because the number of students is small. The simulator seems to be effective to learn behaviors of Ethernet-based networks.**

***Keywords* – Ethernet, Learning computer networks, Computer-network education, Switch simulator, Bridge simulator, MAC address learning.**

I. INTRODUCTION

Ethernet protocol has become much more popular, intelligent, programmable, and important. There have been many "layer-2" protocols, such as ATM or fiber channel. However, many of them has become less popular or used over Ethernet, and Ethernet has gained much more users. Ethernet networks were originally broadcast networks without intelligent network nodes and MAC addresses were completely defined by hard-ware. However, nowadays, Ethernet packets are forwarded by intelligent switches that are comparable to IP routers (and a standardized routing algorithm will also be used [Tou 09]), and MAC addresses may be defined by software; that is, not only addresses for VLAN but also addresses in modern physical. Ethernet interfaces can be replaced by software. Moreover, Ethernet was only used for LAN, but it is also used for wider area, namely, MAN, WAN, and even used globally now.

The upper sub-layer of the Ethernet protocol has become closer to "layer-3" protocols such as IP and comparable to IP because it has become intelligent. Ethernet is usually used in combination to IP as IP/Ethernet and this fact impresses us that whole Ethernet protocol is layer 2 (lower layer) of IP. However, actually, the Ethernet protocol has two sub-layers and the upper sub-layer belongs to network layer (layer 3) although the lower sub-layer of Ethernet protocol based on CSMA/CA is link layer (layer 2). Peterson and Davie [Pet 11] describe Ethernet and IP as comparable protocols for internetworking. (In addition, stacking these protocols causes a difficult duplicated-address problem, which is solved by ARP but causes broadcast storms and further complexity [Kan 12]; however, this is out of scope of this paper.)

Although Ethernet protocol, especially the behavior of switching-based network, has become much more important, the weight of Ethernet switching in computer-network education is still very light, and availability of tools for educating and learning Ethernet is thus also limited. Network simulators are useful for understanding and learning behaviors of network nodes and terminals. Especially, because learning behaviors of a switching network is sometimes complicated, students tend to misunderstand it but an Ethernet simulator helps them. There are several simulators of IP networks for this purpose [Jov 13]; however, Ethernet simulators for learning purpose are very limited. There are various simulation software tools that can simulate Ethernet. For example, OPNET [Cha 99] and *ns*2 [Cir 11] are famous simulators. This type of simulators was used for advanced network classes [Mah 09][Mak 10][Ril 12]. However, most of these simulators stress advanced or exact (microscopic) simulation, and they are not much suited for introductory education and learning.

In the above situation, to use a good tool in a computer network class for novices, the author developed a CLI-based Ethernet simulator. This simulator, which is called CLSim-ue (Command-Line SIMulator UDP version for Ethernet) or CLSim, can display contents of Ethernet packets to send or to receive in terminals and contents of MAC address tables in switches in "real time". CLSim focuses on simple simulation of communication and watching MAC address tables, so the user can easily change the display and learning parameters. Each simulated terminal or switch is a process and they are "wired" by UDP. CLSim is written in Python and, thus, runs on Windows, Macintosh, Linux, and other operating systems. It was used in the university class and the use was evaluated based on a report assignment. The evaluation result is not statistically significant because the number of students is small; however, CLSim seems to be effective to learn behaviors of Ethernet-based networks.

The rest of this paper is organized as follows. Section II describes requirements on Ethernet simulators for computer-network education. Section III describes the design and implementation of the simulator using a simple example.

Section IV describes more advanced usage examples. Section V describes several implementation issues. Section VI evaluates the simulator based on a report assignment to students, and Section VII concludes the paper.

II. Requirements

Four requirements on this type of Ethernet simulators are described in this section. The first requirement is that both terminals and switches must be simulated by using the simulator, and that their behavior must be displayed by a proper method. In addition, the specifications of wiring between the simulated terminals and switches must be easy. A simulated terminal must show the destination and source addresses and contents of packets to be sent and received. A simulated switch should show the content of the MAC address table to show the result of address learning. These values should be displayed in simulated "real time". The displayed values in all the terminals and switches must be synchronously updated.

The second requirement is that the simulated terminals and switches must be able to run on a single computer. Although an environment close to real network is preferable and the terminals and switches to be simulated are different devices, the author decided that the simulated devices must run on a computer because students are not allowed to use many computers and they may use their own computer for simulation. An easy method to fulfill this requirement is to use virtual machines (VMs). However, it is not easy for the available Microsoft Windows machines to install VMs, so another method is required.

The third requirement is that the simulator must run on an environment that every student can easily set up. It must be thus supplied as an executable binary file or by using a widely and easily available execution environment. If a binary file is selected, `.exe` file of Microsoft Windows must be supplied because most students in our university probably use Windows machines. However, an execution environment based on a machine-independent interpreter, such as Java or Python, may be better because a wider range of machine environment can be applicable.

The fourth requirement is easiness of development or low development cost. Because no budget was available for this development, the author himself developed the simulator. In addition, the available development time was very limited. The implementation must therefore be simple even if the usage is constrained.

III. Design and Implementation

A. Outline

To satisfy the first and second requirements, that is, to implement terminals and switches on a single computer and to enable appropriate (real or virtual) communication between them, each terminal and switch is implemented as a process invoked by the user, namely, a command line. If a terminal window (a command prompt in Microsoft Windows) is assigned to each simulated device represented by a process running on the window, this requirement can easily be satisfied. Although it is laborious for a student to open windows as many as required number of simulated devices and to run a command, it is conceptually simple and intuitive.

The simulated devices must be able to communicate each other, and an easy method to achieve this goal is to use inter-process communications using UDP port numbers for distinguishing devices and device ports. Because all the simulated devices run on a computer, an easy way to distinguish the source and destination devices is to use ports of an IP protocol, i.e., UDP or TCP. Because the protocol to be simulated is Ethernet, namely, packet-oriented protocol but not stream-oriented one, UDP is better; that is, each Ethernet packet is simulated by a UDP packet.

This means each device-port is represented by a UDP port and each connected wire is represented by a pair of UDP ports. These two representations and a method for specifying wiring easier are explained in the following.

First, UDP port numbers are used to distinguish both devices and physical ports of the devices. An easy way to assign a port is to use upper three (decimal) digits of the port number (e.g., 11 to 654) as the device number and to use lower two digits as the device port number. For example, device 550 may have ports 0 to 99 (UDP ports 55000 to 55099). In a command that simulates a switch, the destination devices and ports, which are represented by UDP port numbers, are specified by command parameters.

Second, a connected wire is simulated by a pair of UDP ports which represent the simulated device ports. In a command that simulates a terminal, the destination device and port, which is represented by a UDP port number, is specified as a command parameter. The pair of UDP ports, namely, the wire, must be specified by the commands that represent the device ports connected to the wire.

A simulated terminal physically generates a UDP packet that specifies the same source and destination IP address (i.e., the local host address) but different source and destination ports. The UDP payload contains whole Ethernet frame. The terminal physically receives a UDP packet that specifies the port number that represents the terminal. To display the addresses and contents of the output and input packets, the terminal writes them to the standard output. A simulated switch physically receives a UDP packet that specifies one of the switch ports, namely, that specifies the UDP number of the switch port. It simulates the switch behavior and it displays the contents of the MAC address table when required by writing them to the standard output. If the switching algorithm requires sending a packet to a connected device, it physically sends a UDP packet with the port number that corresponds to the device port.

To satisfy the third requirement, that is, to support easy environment for students, the author decided to use Python 3 to program the simulator. The reason was that Python 3 is installed to many PCs in the university. If students decide to

use their own PCs; however, it is probably easy to download and to install it by using an installer if the URL of the installer is told. Java may also have been available but it was not used because the developed program could be used for certain other purposes if they were written in Python. In addition, installing Java may cause a problem because it easily affect overall environment of the PC.

The simulator is a collection of short programs, so it is offered as a public-domain software and available from http://www.kanadas.com/program-e/2014/04/ethernet_-simulator_for_learnin.html or http://bit.ly/1ukTG6Z .

To satisfy the fourth requirement, that is, to reduce the development cost, CLI was chosen for the user interface, although it is easier for novices to use a graphical user-interface (GUI) than a command-line interface (CLI). He believed that an easy-to-use interface could be realized by a well-designed CLI at least for students in ICT-related departments of universities. Students in such departments should learn CLI and use of CLI-based simulator may be a good chance to study CLI.

B. Simple example

To explain the design and implementation by using an example, **Figure 1** shows a simple simulated network. Terminals PC541, PC542, and PC543 are connected through switches SW550 and SW551.

A program named **`term.py`** is used for the terminals. The command is shown in the figure. Physical port 0 of PC542 (i.e., UDP port 54200) is connected to physical port 1 of SW550 (i.e., UDP port 55001). This connection is expressed by command parameters **`--lp`** and **`--rp`** (which mean "local port" and "remote port"). This command specifies a communication between a terminal to another terminal (only one destination), so parameters **`--lm`** and **`--rm`** (which means "local MAC address" and "remote MAC address") specify the source and destination MAC addresses for communication. Packet output can be inhibited by parameter **`--receiveOnly`**. This parameter is convenient for testing one-way communication, which is important to observe learning behaviors. A promiscuous mode may be specified by parameter **`--promiscuous`**. PC543 is also connected to SW550 in the same way, and PC541 is connected to SW551 in the same way.

A program named **`switch.py`** is used for the switches. The command is shown in the figure. Device port 3 or SW550 (i.e., UDP port 55003) is connected to device port 0 (UDP port 55100) of SW551. This connection is expressed by command parameters **`--lp2`** and **`--rp2`**. Because each connection is specified by using different parameters (which must have

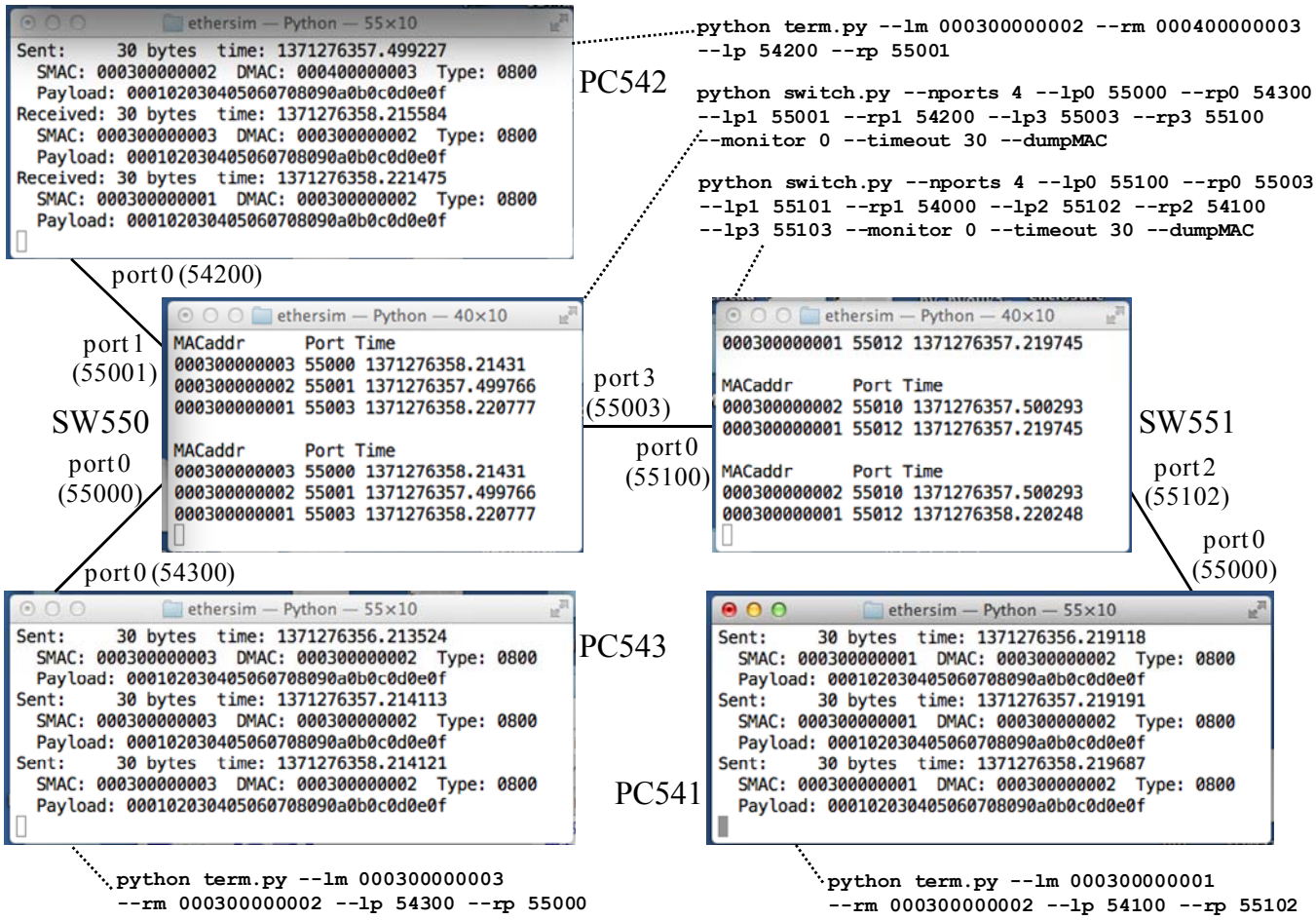


Figure 1. Example of simulator use

different names), the parameter names contain numbers; that is, the parameter names are **lp0**, **rp0**, **lp1**, **rp1**, and so on. The number of ports is specified by a command parameter **--nports**. Parameter **--dumpMAC** specifies displaying the MAC address table, parameter **--timeout** specifies when a learned address is forgotten, and parameter **--monitor** specifies some additional outputs.

The terminal windows show sent and received packets. Because the standard output is used for displaying the packets, the window scrolls rapidly. However, because it repeatedly scrolls by three lines at once and the contents is the same except the timestamp, only the timestamps are effectively rewritten and other contents are easy to be read.

The switch windows show the contents of the MAC address table when **--dumpMAC** option is specified. The contents of this table can also be read easily while they are not changed.

IV. Advanced Examples

More complicated examples than the one shown in the previous section, in which the simulator is more useful, are described as follows.

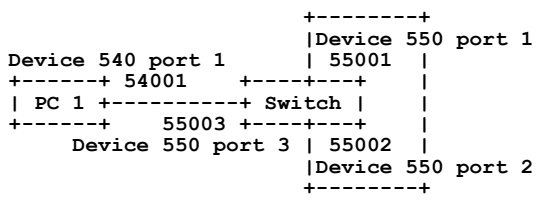
A. Looping

CLSim can be used for simulating behaviors of networks with loops. An Ethernet-based network does not allow redundant links, namely, loops. A simplest loop can be created by connecting two ports of a switch.

```
python switch.py --nports 3 --lp0 55001 --rp0 55002
--lp1 55002 --rp1 55001 --lp2 55003 --rp2 54001
--monitor 1

python term.py --lm 000300000002 --rm 000300000001
--lp 54001 --rp 55003
```

These commands simulate a network as the diagram below.



A loop can also be created by connecting two switches by two wires.

No communication happens by just creating a loop, but, if at least one packet comes from another wire, the loop reproduces packets with the same contents again and again. The packets continue to flow even when all other wires are disconnected; that is, even when all the programs that simulates other packet sources are terminated.

B. Terminal motion

While running a switch (simulator), a terminal connected to the switch may be virtually moved; that is, ther terminal can be reconnected to another switch or to another port of the switch. This live virtual motion, however, requires a special technique. To do so, two scripts (commands) with different ports should be prepared.

```
python term.py --lm 000300000001 --rm 000300000002
--lp 54001 --rp 55001

python term.py --lm 000300000001 --rm 000300000002
--lp 54101 --rp 55101
```

After stopping the execution of the first script, the second script should be started to simulate the motion. The remote port of the destination switch (which has local port 55101, i.e., port 1 of device 551) should be the local port of the second terminal above, i.e., 54101 (port 1 of device 541). This means, the command to invoke the switch is as follows.

```
python switch.py --nports 3 --lp0 50000 --rp0 51000
--lp1 55001 --rp1 54001 --lp2 55002 --rp2 54101
--monitor 0 --timeout 30 --dumpMAC

python switch.py --nports 3 --lp0 51000 --rp0 50000
--lp1 55101 --rp1 54101 --lp2 55102 --rp2 54102
--monitor 0 --timeout 30 --dumpMAC
```

The above configurations makes the communication between the moving terminal and the switches correct; however, they are much complicated. The reason why such a complicated configuration is required is that the destination port of a simulated switch is fixed when it is invoked and the wiring cannot be changed because the wiring is defined by the command parameters. This is different from physical device wiring which can be changed at run time.

C. Timeout

If a terminal moves but it does not generate packets, communication is wrongly disturbed by the past address learning. In the experiment in the previous subsection, the switch learns the new location of the moved terminal immediately because the terminal generates packets. However, if a **--receiveOnly** parameter is specified, the switch does not generate packets and it does not learn the new location of the terminal. The terminal will not thus receive packets that must be sent to it.

For example, a switch learns a terminal connected to switch port 0, and then the terminal is moved to another place (another port or switch). If the moved terminal does not send a packet, the switch sends packet for the terminal to a wrong direction, namely, only to the original port 0. This behavior makes the terminal fail to receive packets. Usually, the terminal will be able to receive packets when a timeout occurs and packets arrived the switch are broadcasted. (A configuration of the timeout time by a **--timeout** parameter makes experiments easier.) However, if no timeout occurs, no new learning occurs unless the terminal sends a packet, so the terminal cannot receive packets.

In addition, more complicated situation that a switch generate packets but they are not learned by a switch that contains temporary wrong table content may be simulated.

V. Implementation issues

Four implementation issues and several minor issues are described in this section.

A. Python versions

Because of incompatibility of Python 2 and 3, it was difficult to unify simulator programs for these versions. The simulator runs on Python 3 on Windows, Macintosh, or Linux as is. If Python 2 processor receives the simulator program for Python 3 it stops compilation because of an error. However, if the statements specified in the programs are rewritten, it can run on Python 2 too.

B. Multi-computer environment

The simulator in the current form runs only on a single computer, but it can be easily extended to multi-computer environment. All the terminals and switches simulated by the simulator are assumed to run on a single PC because the author assumed that no student wants to run the simulator in multi-computer environment. However, because UDP/IP is used for communications between simulated terminals and switches, it is easy to adapt the simulator to a multi-computer environment. Instead of specifying only a UDP port, if the program is slightly modified so that a an IP address and a UDP port are specified in a command parameter for a source or destination port, the simulator works among different computers.

C. Complexity of wiring

There are three reasons that causes wiring between simulated devices complicated. The first reason is that the wiring in the simulator is uni-directional. A physical Ethernet wire is usually bidirectional even when the communication is half duplex. Devices connected by a simulated wire in the simulator must however specify the source and destination ports separately because the commands that implements the devices runs independently. Therefore, if the parameters are inconsistent, the simulation fails.

The second reason of complexity is use of UDP port. Because all the simulated devices must run on a computer, each simulated physical port must have different UDP port number. In the standard usage, the UDP port number combines the physical port number and the device number. The author believes this usage is conceptually simple, but still it complicates the specification of wiring.

The third reason of complexity is caused by the method of specifying command parameters. Because all the command parameters must have different names, each port must be distinguished by the parameter name, namely, **`--lp0`** and **`--rp0`**, **`--lp1`** and **`--rp1`**, and so on. Because the postfixes 0, 1, … are independent from the device port numbers, it is quite complicated.

To solve this complexity problem, an additional program (i.e., a script generator) was developed. The reason of this development was that it seemed difficult for the students to configure a network consistently, although he had believed that the standard usage of the simulator was simple enough for them. The script generator input a specification of a network such as follows.

```
switch 510
  linkto 410   # server
  linkto 511   # switch
  linkto 512   # switch

switch 511
  linkto 411   # PC 1

switch 512
  linkto 412   # PC 2

# Server 0
terminal 410
  localMAC  000300000001
  remoteMAC 000300000002

# PC 1
terminal 411
  localMAC  000300000002
  remoteMAC 000300000001

# PC 2
terminal 412
  localMAC  000300000003
  remoteMAC 000300000001
```

The script generator generates scripts for specified terminals and switches. The above specification contains three terminals and three switches. Each simulated wire, which is bidirectional, is specified only once, and no command names are specified. Physical port numbers are also generated automatically.

Although the script generator was distributed to students, it was after the report deadline. None of them thus used it in this year.

D. Busy wait on Windows XP

The simulated terminal program generates a packet per second. A busy wait was used in the original program to wait for a second. This program works well on newer Windows (Windows Vista, 7, and probably 8), Macintosh, and Linux. However, students found that it did not work well on Windows XP because a busy wait spent too much time. The program was not tested on Windows XP because this OS was not available for the author. It was rewritten to sleep while waiting. The wait time must be 1 ms or more because a smaller time specified is regarded as 0 on Windows XP.

VI. Evaluation

The simulator was used for an assignment of a report on Ethernet. The students of a computer network class were asked to hand two reports. The first report should describe the design and rough behavior of an Ethernet-based network manually, and the author guaranteed to give full score for this report if it was handed (submitted). The students were encouraged to use the simulator when writing the second report. To use the simulator was originally a must; however, because the author found that it was difficult for them to use the simulator, he loosened the condition. Students who used the simulator got additional marks but simulator use was not a must.

Several additional conditions on this report assignment are described below. The class is for fourth-grade night-class students of the Department of Informatics, Communication and Media, but first- to third-grade students and several students from other departments also took this class. The average knowledge and skill of the students are thus rather

Table 1. Result of report assignment

Item	Number of students	2nd report score
1st report submission	21	-
2nd report submission	13	61%
Using the simulator	9	67%
Not using the simulator	4	48%

low. Most of the students who tried the simulator used their own Windows PC instead of using PCs in the University probably because it was not convenient for them to use the PCs there. Although the simulator can run on Macintosh or Linux, all of them used Windows. In addition, some (many?) of them still used Windows XP instead of a newer versions of Windows.

The results of this report assignment are summarized in **Table 1**. The numbers of submitted first and second reports are 21 and 13. In nine of the 13 second reports the simulator is used. (Initially, the number of submitted reports was small, but the author encouraged to submit it after the deadline.) The average percentage of the second report (without additional marks) using and not using the simulator are 67% and 48%. There seems to be a difference in these percentages. This difference may be cause by the effect of using and learning from the simulator or by knowledge and skill of the students, which are not acquired by the simulator. However, because the number of samples is small, the difference is not statistically significant.

Several students (maybe four to ten students) seemed to try but to abandon the simulator because of several reasons. The major reason is that the difficulty of wiring terminals and switches for the simulator; that is, it was difficult for them to prepare parameters of switch and terminal commands consistently. However, there are also several other reasons. One reason is that some students did not know how to use the CLI (i.e., the command prompt of Microsoft Windows). Especially, it was difficult for some students to set up the current directory and the path for executing Python. Another reason is that the simulator was distributed by a zip file but at least one student did not know how to decompress it (probably because a zip file looks like a normal folder in Windows).

Other reasons of failures are as follows. At least two students used the simulator but did not obtain correct results. The reason may be the difficulty of wiring devices. At least one student failed to install Python, but the reason was unknown because she did not know the reason and the author could not see the reason either because it was her own PC. At least one student correctly used the simulator but he seems failed to understand switch behavior. The author found it because his score of the final examination were very low. However, most of the students who used the simulator seem to understand Ethernet better.

In this report assignment, the author did not take time for practice in the class time. It may have got much better results if the usage of the simulator was taught in a computer room.

VII. Conclusion

A CLI-based Ethernet simulator that can display contents of Ethernet packets in terminals and contents of MAC address tables in switches in "real time" was developed for a university class. Each simulated terminal or switch is a process and they are "wired" by UDP. The simulator, which is in public domain, is written in Python and, thus, runs on Windows, Macintosh, Linux, and other operating systems. This simulator was used in the university class and the use was evaluated based on a report assignment.

The evaluation result shows the average score of reports that were written by using the simulator was much better, but it is not statistically significant because the number of students is small. Although the simulator used for this evaluation still had several problems to be solved, the simulator seems to be effective to learn behaviors of Ethernet-based networks and it seems to be promising if the above problems are solved.

References

[Cha 99] Chang, Xinjie, "Network simulations with OPNET", *31st Conference on Winter Simulation* (*WSC'99*), pp. 307−314, 1999.

[Cir 11] Ciraci, S. and Akyol, B., "An Evaluation of the Network Simulators in Large-Scale Distributed Simulations", *1st International Workshop on High Performance Computing, Networking and Analytics for the Power Grid* (*HiPCNA-PG'11*), pp. 59−66, 2011.

[Jov 13] Jovanović, N., Jovanović, Z., Popović, O., Stanković, I., and Zakić, A., "Computer Network Simulation and Visualization Tool", *11th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services* (*TELSIKS 2013*), October 2013.

[Kan 12] Kanada, Y. and Nakao, A., "Development of A Scalable Non- IP/Non-Ethernet Protocol With Learning-based Forwarding Method", *World Telecommunication Congress 2012* (*WTC'12),* March 2012.

[Mah 09] Maheswaran, M., Malozemoff, A., Ng, D., Liao, S., Gu, S., Maniymaran, B., Raymond, J., Shaikh, R., and Gao, Y., "GINI: A User-level Toolkit for Creating Micro Internets for Teaching & Learning Computer Networking", *ACM SIGCSE Bulletin*, Vol. 41, No. 1, pp. 39−43, March 2009.

[Mak 10] Makasiranondh, W., Paul Maj, S., and Veal, D., "Pedagogical Evaluation of Simulation Tools Usage in Network Technology Education", *World Transactions on Engineering and Technology Education*, Vol. 8, No. 3, pp. 321−326, 2010.

[Pet 11] Peterson, L. L. and Davie, B. S., "Computer Networks, Fifth Edition: A Systems Approach", Morgan Kaufmann, 2011.

[Ril 12] Riley, G. F., "Using Network Simulation in Classroom Education", *2012 Winter Simulation Conference* (*WSC'12*), 2012.

[Tou 09] Touch, J. and Perlman, R., "Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement", RFC 5556, IETF, May 2009.