Network Tester: A Generation and Evaluation of Diagnostic Communication in IP Networks

J. Pullmann and D. Macko
Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava, Slovakia
dominik.macko@stuba.sk

***Abstract*—There are many active intermediary devices in computer networks. The network has to be verified whether it functions as it should – i.e. whether some intermediary device in not misconfigured. It is a difficult task, because various scenarios must be tested, which takes time. Fortunately, there are tools that help to automate this process. However, the existing tools are mostly oriented towards generation of specific periodic traffic for network performance analysis, and it is difficult to use them for functional verification of the network. Therefore, we propose in this paper a new method, along with the corresponding tool, which is targeted towards generation of certain traffic and analysis of the network response to verify its function. The primary focus of the method is for testing of local area networks, which usually consists of switch devices. The experimental results proved that the proposed tool is usable to verify specific functional aspects of the switched networks. The tool can be used in education to verify students-configured local networks or to test software switches implemented by students.**

I. Introduction

The modern computer networks (based on IP – Internet Protocol) usually consist of many intermediary devices, such as switches, routers, firewalls, load balancers, etc. These devices can be manageable, meaning that they can be configured to adjust their function to some user requirements. Study of the network devices configuration is a specific area in education. Students configure network devices to perform some function and a teacher must check whether they configure them correctly. This is quite time-consuming task.

Moreover, as there is a high focus on network virtualization nowadays, many functions of intermediary devices are done in software. Therefore, students often are tasked to develop software programs that perform specific functions of some hardware device (e.g. switch function). Students must connect their software implementation to a real network to verify the function, and they do not always have manageable network devices available at home, which limits their flexibility to debug their solutions. Also, verification of such implementations by a teacher takes just too much time and there is a need to automate this process.

This work is therefore focused on development of a method and corresponding tool that will be able to quickly verify some function of a network device, or a portion of a switched network. The proposed tool enables a teacher to create test scenarios, according which the tool generates network traffic, and the tool analyzes the response. Based of specification of expected response, the tool is able to determine whether the test scenario passed or failed. Such a tool helps the students to verify configured hardware devices or to continuously verify software implementations of a network function during development, and it also helps teachers to evaluate final students' implementations of software network functions.

The following section describes some analyzed existing works targeting generation and analysis of network traffic. Section 3 contains the proposed new method for generation and evaluation of diagnostic communication in IP networks and describes the developed tool that implements the proposed method. In Section 4, we mention verification of the tool and experimental results. And in Section 5, we discuss the conclusions of our work and give information about further work.

II. Related Works

There exist many tools that enable generation of some testing traffic in computer networks. They differ by several parameters, such as complexity, functionality, user interface, operating systems support, or licensing. One can easily find a comparison of some popular tools, such as in [1-4]. The main goals of network-traffic generation are to test the performance and behavior of the network. For example, such tools can be used to test network firewall, measurement of network bandwidth, network device overloading, or simulation of a network virus. In this section, we analyze some of the popular tools.

OSTINATO [5] is a tool for generation and analysis of network traffic. It offers an easy-to-use user interface, controlled either via a GUI (Graphical User Interface) or via Python API (Application Programming Interface), which enables automation of network testing. It is an open-source free-to-use tool that supports multiple platforms (Windows, Linux, BSD, or Mac OS X). It enables to modify any value in any stateless network protocol, concatenate packets in any order, and thus create specific traffic streams. It can be used for network load testing or functional testing. From the analyzing point, it offers statistics and captures incoming traffic; however, it does not differentiate generated and standard traffic in the network. Further analysis must be done manually by the user, or by external application. Therefore, it is hard to use for evaluation of a network device function (hardware or software. In Fig. 1, samples of OSTINATO graphical user interface are provided. On the left, we can see some obtained network statistics. On the right, a portion of

This is an accepted version of the published paper:
J. Pullmann and D. Macko, "Network Tester: A generation and evaluation of diagnostic communication in IP networks," 2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, 2018, pp. 451-456.
doi: 10.1109/ICETA.2018.8572067
URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8572067

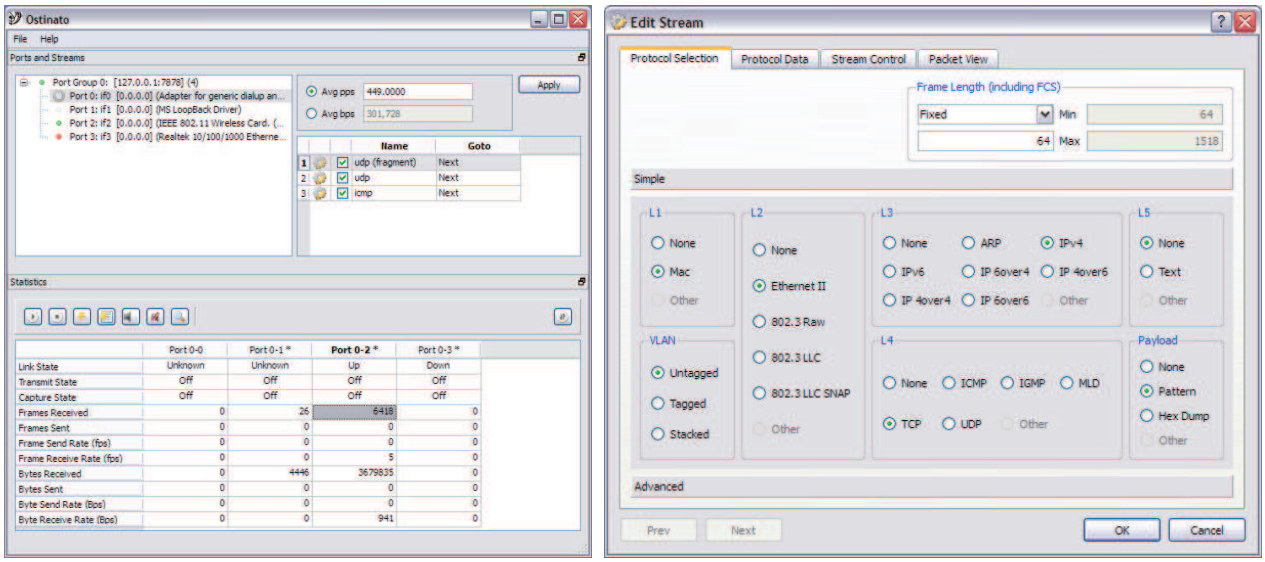

Figure 1. OSTINATO graphical user interface [5].

editing possibilities is illustrated, where a user can select a protocol for individual layers of the protocol stack – from the physical layer to the transport layer. Application layer can be simulated by a text.

Another tool, iPerf [6], offers active measurement of maximal bandwidth in IP networks, reporting bandwidth, loss, or other parameters for each test. A user can control the tool via a CLI (Command Line Interface); however, there exist a GUI extension called jPerf. Similarly to OSTINATO, iPerf is also an open-source, free-to-use, multiplatform tool. In comparison to previous tool, it offers fewer options to configure the traffic stream. Except the measured statistics, there is no analysis or evaluation of network function offered; thus, it is unsuitable for our purpose.

The PackETH [7] multiplatform tool offers high configurability of generated network streams, similarly to OSTINATO. It can be controlled by a CLI as well as a GUI. It is primarily focused on Ethernet networks. It enables to save the test scenario, and afterwards to load it. The advantage is that the generated network stream is in the pcap form, which means that the captured traffic by another tool (such as Wireshark [8]) can be replicated. On its own, PackETH does not support analysis of the network traffic, thus the test cannot be automatically evaluated. Thus, it is unsuitable for network function verification.

GDC tool [9] is a tool for generation and evaluation of network traffic. Its function is the closest from the existing tools to our purpose. However, this tool was primarily developed to test function of a router device, which means it is very limited on type of traffic that it can generate and analyze (ICMP, ARP, RIP). It uses a configuration file in an XML (eXtensible Markup Language) form, which specifies a test scenario to be generated and evaluated. This is rather unsuitable for creation of new test scenarios, since the tool does not offer a proper configurator

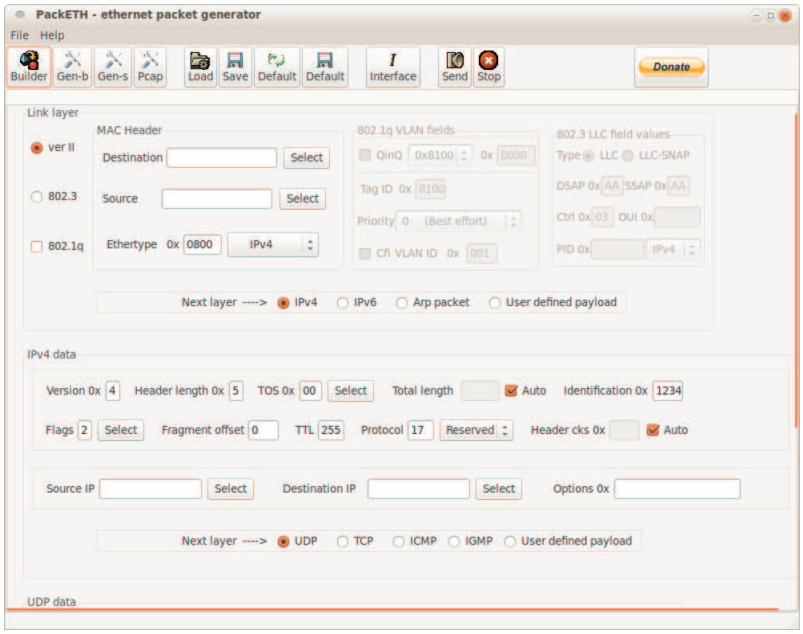

Figure 2. PackETH graphical user interface [7].

graphical interface. However, some functions and ideas used in this tool can be used as an inspiration for development of a new generation and evaluation tool.

The analyzed existing tools do not fulfill requirements of our intended usage, mentioned in Section 1. The existing traffic generators offers high configurability; however, they are too complex and thus complicated to use, and they lack features on the evaluation part. They would need to be used in conjunction with some network analysis tool, such as Wireshark [8] or KaTaLyzer [10], accompanied by manual evaluation. Therefore, we have decided to propose a new traffic generation and evaluation automation method, which is more closely described in the following section.

III. The Proposed Network Tester Method

In order to test specific functionality of a network device or some portion of the network, we have stated the following requirements for a new automation method:

- Easy configuration of the test traffic (i.e. the diagnostic communication for a local area network) by a user.
- A test scenario should consist of various communication flows.
- Generation of the configured test traffic to the network.
- Analysis of the captured communication from the network.
- Evaluation of a test scenario after traffic generation, capturing, and analysis.
- Providing statistics and control messages.
- Saving and loading the configured test scenarios to and from a file.
- Simple and easy-to-use user interface of the implemented tool.
- Modularity and extensibility of the implemented tool.

These requirements represent a basis for our proposal, which intends to meet all of them. The result from the automate evaluation must be obtained in a very short time, in order to be beneficial for students as well as teachers.

A. Method Overview

The proposed tool will operate on a single device (e.g. personal computer, laptop, or microcomputer) with two network interface cards (NICs). Both NICs will be connected to the local area network to be tested (or to a single network device to be tested). A condition is that the NICs can communicate with each other via the local network. One of the NICs will represent an initiator of the diagnostic communication according to a test scenario created by a user (a teacher). The second interface will capture the incoming communication, and respond to the traffic in some cases to simulate a real communication. Since this behavior is centralized in a single device and a single tool, it knows what the test scenario was and thus it can evaluate the result of the test. In Fig. 3, an overview of the physical architecture connecting the PC, running the proposed tool, to the network under test is illustrated.

The proposed testing method consists of four modules: task creator, scenario manager, test executor, and results analyzer. These are illustrated in Fig. 4 and further explained in the following text.

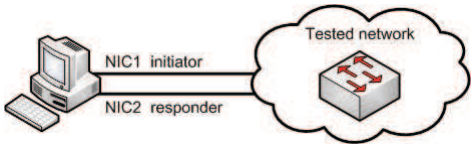

Figure 3. An overview of the physical architecture used for testing.

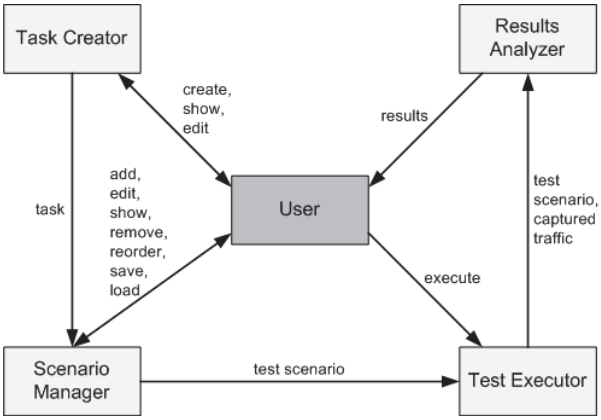

Figure 4. An overview of interactions between the modules.

1) Task Creator

This module is used to create test tasks by a user, and also to show or edit the created tasks. A test scenario can contain multiple tasks (e.g. ARP request simulation, loading of a web page using HTTP, and downloading a file using FTP). The user is able to select various parameters of the task to be executed (i.e. the corresponding traffic is generated), such as selection of a network protocol, modification of header contents, or specification of a number of packets to be sent in the task. By creation of individual tasks, the user defines a test scenario as needed.

2) Scenario Manager

This module is used for management of a test scenario, which enables manipulation with the scenario tasks (adding, editing, showing, removing, duplicating, and reordering). All of these give a user flexibility to create a suitable scenario or to modify the existing one in a short time. The existing test scenarios can be saved for further usage. This module also enables to load a saved scenario for modification or execution, and it also enables to delete the whole scenario. This module also serves for specification of which port is used as an initiator and which as a responder of the test communication.

3) Test Executor

This module serves for execution of individual tasks according to the selected test scenario. It is used for traffic generation, receiving (capturing for further analysis), and responding. Since the testing network traffic runs in both directions, it is inevitable to be able to send and receive traffic at both NICs.

4) Results Analyzer

This module analysis, evaluates, and provides results of the test based on generated and received network communication. Output of the module contains traffic statistics, such as a number of sent or received packets,

and evaluation, whether the network communication has behaved in individual test scenarios as expected. For example, a user wants to verify whether a network firewall denies HTTP traffic but allows all other communication. If the generated HTTP traffic by the initiator NIC is received by the responder NIC, the test scenario is evaluated as failed (unsuccessful), because it was expected that it will be blocked by the firewall. The user is then notified which specific task has failed in the test scenario.

B. *Network Tester Implementation*

The proposed method was implemented into a tool called Network Tester in the Java programming language. For operations with the network interfaces, the jNetPcap library was used. The Jackson library was used for working with data in the JSON (JavaScript Object Notation) format.

A user interacts with the tool using the implemented GUI, which consists of four main windows: Port Settings, Scenario Management, Task Creator, and Scenario Results.

Port Settings is the first window displayed after starting the tool (shown in Fig. 5). It serves for selection of the two network interfaces, which will be used by the tool, from all network interfaces the device has. In order to fully utilize the tool, the device has to provide at least two network interfaces. Otherwise, the tool displays a message about minimal amount of required NICs and does not enable the user to execute a test. Other functionality, which does not require NICs interactions, is enabled, such as test scenario management or task creation.

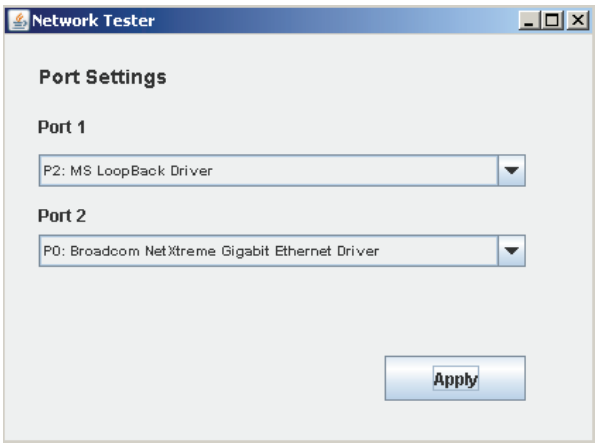

Figure 5. Port Settings window.

Scenario Management is the key window enabling control of the tool. It enables user to interact with the previously described Scenario Manager module. The main purpose is to manage a test scenario, which is represented by a list of test tasks in the main part of the window (shown in Fig. 6). The top menu contains three items: File, Settings, and Help. The File item enables to save scenario, load a saved scenario, and erase the loaded scenario. The Settings item enables to modify the previously selected port settings. And the Help item enables to display the user guide, if required. The buttons enable to control individual functions of the Scenario Manager module. Most of them require selection of a task in the list of the test scenario to action with that task. The Task Details button shows detailed parameters of the selected task. The Create Task button opens the Task Creator window. The Edit Task button also opens the Task Creator window; however, the parameters of the task are filled according to the selected task. The user can modify them and save the changes. The Delete Task button removes the selected task from the test scenario. The Duplicate Task button copies the selected task and adds it at the end of the list. The user can then modify the parameters of the new task, which makes creation of a new task with similar parameters to the duplicated task very fast. The Move Up and Move Down buttons moves the selected task in the list (reorder the tasks), which enables to adjust the test scenario. The Run Scenario button executes the current scenario.

Task Creator (shown in Fig. 7) is a window that enables to create new tasks, which will be added to the current test scenario. It contains multiple text fields for task parameters, according to which the network traffic will be generated. The Source Port parameter specifies the NIC that will be the initiator for that task. The Type parameter enables to select protocol (UDP, TCP, ICMP,

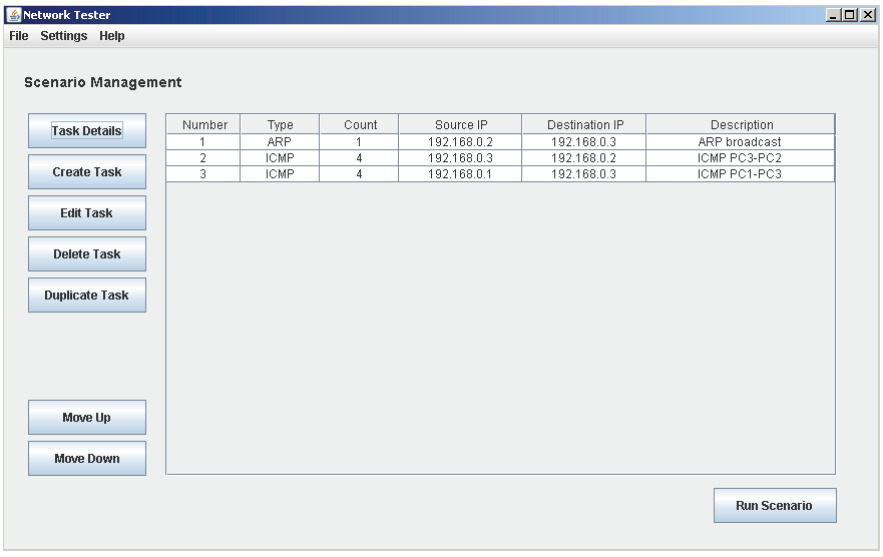

Figure 6. Scenario Management window.

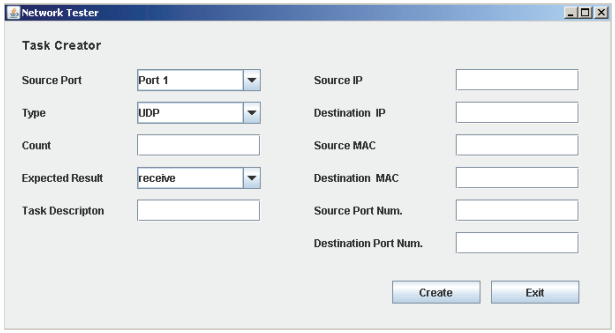

Figure 7. Task Creator window.

or ARP), for which the traffic will be generated. Count specifies how many messages will be sent. The Expected Result parameter specifies whether the message should or shouldn't be received at the responder NIC. There is also a field for a unique short description of the task, in order the user to know what the task intents to do. Other text fields are used for specification of addresses and ports to be used in the generated communication. In case some field contains erroneous value, a message reports the issue to the user.

Scenario Results (shown in Fig. 8) provides both summarized and detailed information about the test scenario results. It also shows progress of testing during the test execution. Thus, the user is informed which task is currently executed and what the result of finished tasks is. The user is also informed when the test is completed (i.e. all tasks from the test scenario are finished). In the Test state text box, basic information is provided, such as test progress, number and percentage of successfully completed tasks, tasks' ordinal numbers and tasks results (OK or Failed). In the Results table, several basic parameters for each task are provided along with the test result and statistics. There is a possibility to save results to a file in the JSON format. The Task Details button enables to show all configured parameters of the selected task in the table.

As already mentioned, the current test scenario can be saved into a JSON-formatted file. The saved test scenario contains a list of test tasks. For each task, name and value


```
▾ {
    taskId: 2,
    type: "TCP",
    sourcePort: 2,
    count: 1000,
    expectedResult: "receive",
    srcPortNumber: 55000,
    dstPortNumber: 80,
    srcIp: "192.168.2.1",
    srcMac: "a2:aa:aa:aa:aa:aa",
    dstIp: "192.168.2.99",
    dstMac: "c2:cc:cc:cc:cc:cc",
    taskDescription: "PC1 na web server"
},
```


Figure 9. An example of a task saved in a JSON file.

of all of its parameters are provided, such as type, count, description, or destination IP address. An example of a task saved into such a file is provided in Fig. 9.

IV. Experimental Verification

The implemented functions in the tool have been verified continuously during the tool development. Partial functionalities of each module have been verified separately, and also, the tool has been verified as a whole.

During the development, the verification was accomplished using the network emulator GNS3. Final version of the tool was verified with the use of real hardware and software switch devices. During verification, we have used the Wireshark packet analyzing tool and checked whether the captured traffic corresponds to the results obtained by the Network Tester tool.

Three test scenarios have been used for experimental verification. The first one was generating small amount of packets to verify the basic functionality of the tool. The second one contained more complex tasks, generating higher amount of traffic to test filtering options of a student-implemented software multilayer switch. The third scenario was created to test robustness of the tool, generating a very high amount of traffic (tens of thousands packets).

Even in the last high-load test scenario, the tool was able to obtain the result in a couple of minutes. Thus it is very fast, helping students and teacher to speed-up the evaluation processes.

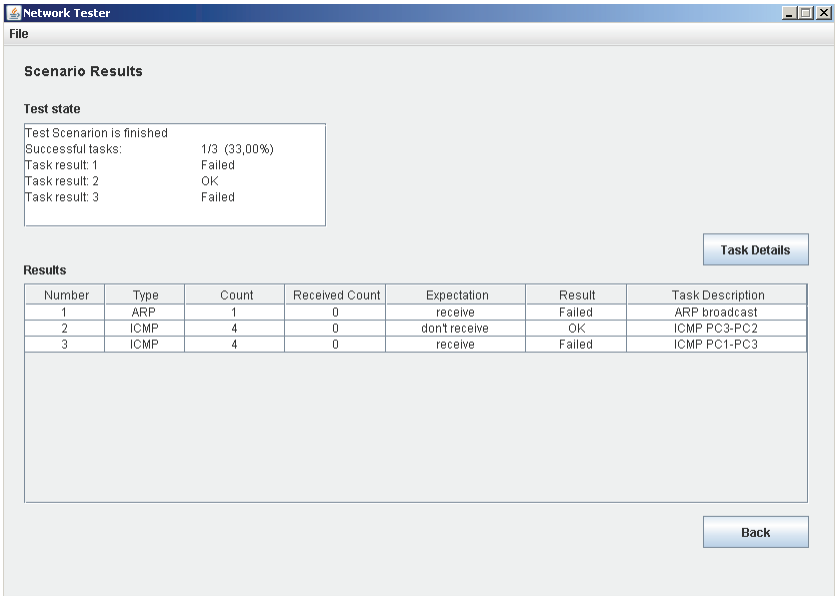

Figure 8. Scenario Results window.

V. Conclusions

The work described in this paper was focused on development of a method and the corresponding tool that will be able to quickly verify some function of a software or hardware switching network device, or a portion of a switched network. The proposed tool, named Network Tester, enables a teacher to create test scenarios, according which the tool generates network traffic and analyzes the response. Based of specification of expected response, the tool is able to determine whether the test scenario passed or failed, giving enough information to evaluate whether the tested device or network functions properly.

Such a tool can be used by students to verify configured hardware devices or to continuously verify their software implementations of a network function during development. The tool also helps the teachers to quickly evaluate students' final implementations of software network functions, assigned as a semestral works.

The proposed evaluation automation method and tool has been verified using real student's works and it serves its purpose well. It can be further extended for support other protocols to verify other network functions, such as the basic function of a routing device.

Acknowledgment

This work was mainly supported by the Slovak Cultural and Educational Grant Agency (KEGA 011STU-4/2017 – "Update of computer networks curricula based on needs of practise") of the Ministry of Education, Science, Research and Sport of the Slovak Republic. It was also partially supported by the Slovak Scientific Grant Agency (VEGA 1/0836/16) and the Slovak Research and Development Agency (APVV-15-0789).

References

[1] A. G. Patil, A. R. Surve, A. K. Gupta, A. Sharma, and S. Anmulwar, "Survey of synthetic traffic generators," in *2016 International Conference on Inventive Computation Technologies (ICICT)*, 2016, pp. 1–3. doi: 10.1109/INVENTIVE.2016.7823282

[2] S. Mishra, S. Sonavane, and A. Gupta, "Study of traffic generation tools," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, no. 6, pp. 4–7, 2015.

[3] S. S. Kolahi, S. Narayan, D. D. T. Nguyen, and Y. Sunarto, "Performance monitoring of various network traffic generators," in *2011 UkSim 13th International Conference on Computer Modelling and Simulation*, 2011, pp. 501–506. doi: 10.1109/UKSIM.2011.102

[4] S. M. Wong, *An Evaluation of Software-Based Traffic Generators using Docker*. Stockholm, Sweden: School of Computer Science and Communication, KTH, 2018. Master thesis.

[5] P. Srivats, "OSTINATO: Network Traffic Generator and Analyzer," 2017. [Online]. Available: http://ostinato.org.

[6] "iPerf - The ultimate speed test tool for TCP, UDP and SCTP," 2016. [Online]. Available: https://iperf.fr/.

[7] "PackETH," 2017. [Online]. Available: http://packeth.sourceforge.net/packeth/Home.html

[8] U. Lamping, R. Sharpe, and E. Warnicke, "Wireshark User's Guide: For Wireshark 2.1", 2014. [Online]. Available: https://www.wireshark.org/download/docs/user-guide-us.pdf.

[9] M. Procházka, D. Macko, and K. Jelemenská, "IP networks diagnostic communication generator," in *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2017, pp. 1–6. doi: 10.1109/ICETA.2017.8102520

[10] T. Kovacik, I. Kotuliak, and P. Podhradsky, "Real-time traffic analysis in Ethernet," in *15th International Conference on Systems, Signals and Image Processing (IWSSIP 2008)*, 2008, pp. 69–72. doi: 10.1109/IWSSIP.2008.4604369