IP Networks Diagnostic Communication Generator

M. Procházka, D. Macko and K. Jelemenská
Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava, Slovakia
dominik.macko@stuba.sk

***Abstract*—Generation of diagnostic communication (GDC) is an important task in the area of computer networks that is used in various circumstances. For example, it is used for the purpose of testing networking hardware, such as routers or switches, for analyzing and benchmarking of networking systems or web services. Although there are various solutions using GDC, typically, they are specialized for generation of either general network traffic, network communication for diagnosis of anomalies in the network, or aimed on analysis of the captured network packets. Therefore, these solutions are not suitable for functional verification of network devices. In this work, we concentrate our research on the functional verification of software routers. We introduce the GDC diagnostic tool for IP networks that is dedicated to validation of software routers' functional correctness. This tool is based on GDC, driven by a configuration file and it provides also the response communication capturing and evaluation of the executed tests in a visual form. Although it was mainly intended to simplify the basic functionality validation of the software routers implemented by students during the coursework, it can also be useful to verify the functionality of the real network devises.**

***Keywords*—ARP; diagnostic communication generation; IP networks; software routers functional verification; RIP**

I. Introduction

At the Faculty of Informatics and Information Technology, there are many courses in which one or more hands-on assignments are required to complete for students to get the necessary knowledge and practical experience. The work of the teachers in these courses is to check the correctness of these assignments, which is often very demanding, especially in terms of time. Also, students do not have an easy access to the hardware devices and therefore cannot adequately verify the functionality of their solutions outside the laboratory.

WAN technologies course is no exception. One of the hands-on assignments in this course is the design and implementation of a software router that is able to create an ARP (Address Resolution Protocol) table and a routing table, it can perform static paths routing, as well as routing using RIP (Routing Information Protocol). Testing for the correctness of the assignment solution therefore consists of several activities or scenarios for each of these functions, which need to be evaluated.

The aim of our research is to speed up and streamline the process of evaluating these tasks and to develop a tool that will automate this testing. Based on the scenarios to be performed, the instrument should carry out tests, gain and then display the results of the tests that demonstrate the success of these tests.

However, active networking devices, such as routers or switches, represent essential elements of almost every computer network. With the expansion of the Internet, the demands of network hardware market were substantially increased and the new vendors came along. Some of these vendors are focused on modular networking hardware development. It means, the networking hardware is composed of a generic motherboard with several network interfaces and the required functionalities of this networking hardware are obtained by a suitable software solution installed on the motherboard.

This configuration of the networking hardware enabled to create several open source projects, such as DD-WRT, Tomato, OpenWRT, M0n0wall, PfSense, Vyatta [1] etc. The main focus of the mentioned projects is to allow a user to adjust the networking hardware to user-specific, most detailed requirements.

A great amount of testing comes with the adjusting or creating of the new software for this networking hardware. Therefore, in order to support this emerging configurable networking hardware, the proposed tool should also be utilizable for testing real hardware routers. When changing configuration parameters or minor changing of the implementation, the tool should also be used to test other network elements (e.g. switches).

As a result, we are introducing a tool for generation of diagnostic communication (GDC), which allows an automatization of validation of the router's functionality. This tool will be also available for students, who are implementing the software router.

The paper is organized as follows. In Section 2, the existing GDC methods and tools are summarized. The proposed new automatization tool is introduced in Section 3. The proposed solution verification and testing is discussed in Section 4. Finally, the results are summarized and the plans for future work are provided in the concluding section.

II. Related Work

Concerning network traffic there are plenty of solutions available [2] that are able either to generate network traffic or to capture and analyze the network communication. Some of them represent open-source solutions (e.g. [3] or [4]), the others are either closed or commercial tools (e.g. [5]). Five of the available tools were selected and they are shortly summarized in Table 1.

TABLE I.
EXISTING NETWORK TRAFFIC GENERATORS AND ANALYZERS

Tool	Description
Harpoon [3]	Flow-level traffic generator. Harpoon can be used to generate representative background traffic for application or protocol testing, or for testing network switching hardware.
KaTaLyzer [6]	Network traffic analyzer for Linux based operating systems (routers, servers and desktops). It supports widely used protocols (Ethernet, IP, TCP, UDP, HTTP, SSH, SIP, etc.).
NetScanTools ® Pro [5]	Tool for generating TCP, UDP, ICMP, ARP and RAW protocols packets with option of modifying headers.
OSTINATO [4]	Open-source, cross-platform network packet crafter/traffic generator and analyzer with a friendly GUI. It crafts and sends packets of several streams with different protocols at different rates.
Wireshark [7]	Wireshark is the world's foremost and widely-used network protocol analyzer.

These were used as an inspiration and they helped us to specify the requirements for the novel GDC tool that would be suitable for the intended usage.

The most interesting features of Harpoon [3] are the client-server architecture and the configuration file, which provides scenarios for the traffic generation process. Another interesting feature is its self-configure function, which means the TCP (Transmission Control Protocol) communication generation can be configured automatically based on the captured traffic analysis. However, for the intended usage, this feature would not be very useful. In multiplatform commercial tool called OSTINATO [4], the Protocol Builder is internally used along with exclusive Ethernet port privileges, which enables to check the whole data flow through the ports. On the other hand, only the basic statistics about the network ports traffic are implemented in this tool, which is not sufficient for our purpose. In terms of modularity, NetScanTools ® Pro [5] consist of modules, which can be also modified. It is an interesting solution where a module represents the type of scenario and a configuration file represents the scenario set-up. Analysis tools (Wireshark [7], KaTaLyzer [6], and a part of NetScanTools® Pro) have the most relevant features concerning visualization, filtering, and a graphical user interface.

Despite of all the interesting features of these tools, they have not been designed for the network devices validation purpose. Therefore, it will still be difficult to adopt them for this purpose. In addition, the available tools are typically quite robust and complicated, and thus less suitable for education purpose. These were the reasons why we have concentrated our work on novel GDC tool development that will be focused mainly on parts with the most significant asset for the intended usage. However, the analyzed and tested solutions inspired some of the features of the newly proposed tool, concerning architecture, language selection, user interface and other aspects. For better understanding of GDC, it was appropriate to divide the whole problem into two sub-problems, namely network traffic generation and network traffic analysis.

III. THE NOVEL GDC TOOL DESIGN

This section summarizes the novel GDC tool design, specified requirements for the tool, its architecture, and the graphical user interface.

A. Requirements specification

Based on the analysis of available solutions and consequent testing of selected subset of them, the following requirements for the novel GDC tool development have been specified. They are divided into two groups: functional and non-functional requirements. These requirements have to be met during the tool development process for correct functionality of the proposed solution.

Functional requirements for the GDC tool include:

- handling the configuration file,
- generating packets based on the configuration file,
- selection of the two network interfaces,
- generating of and replying to ICMP (Internet Control Message Protocol) messages,
- generating of and replying to ARP requests,
- generating of RIP messages,
- visualizing the traffic statistics, captured packets, and passed tests.

Non-functional requirements for the GDC tool include:

- simplicity,
- effectiveness,
- modularity, and
- usability.

B. Architecture

The proposed architecture of the new GDC tool consists of two main modules: client and server. These modules operate on the same machine (e.g. a personal computer) with two Ethernet interfaces as shown in Fig. 1.

The server side, representing the generation part (GP), is mainly in charge of handling configuration files and creating packet streams. Packets are sent through one of the interfaces afterwards.

The client side, representing the analysis part (AP), receives packets (going through the network including

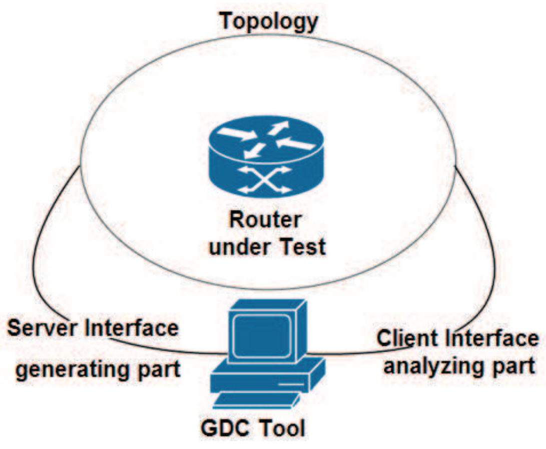

Topology
Router under Test
Server Interface generating part
Client Interface analyzing part
GDC Tool

Figure 1. The adopted testing scheme

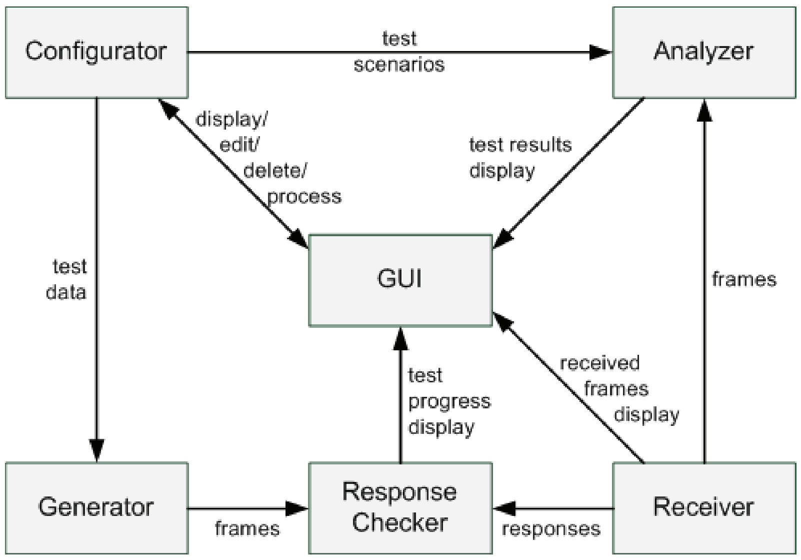


Figure 2. The System Architecture

device under test) on the second interface of the machine, analyses the received packets, and evaluates the results of testing.

Both parts consist of several modules. The module that covers both GP and AP is a graphical user interface (GUI) that displays information from individual modules and contains tool controls. The co-operation of the individual tool modules is summarized in Fig. 2.

The solution of a single-machine client-server application has been selected in order to simplify the diagnostic access. The user does not need to dispose with several machines; instead, he/she can operate with both parts easily. However, client and server use separated network interfaces that are dedicated and strictly controlled by one of them. The modular solution meets the criterion listed in the non-functional requirements, which is the modularity of the tool and also its simple extensibility. Let us introduce the two parts modules in more detail.

GP modules:

- Configurator – enables creating, editing and deleting configuration files, and performing the tests listed in a specific configuration file. This module is also sending the test data to the analyzer so that it can evaluate the tests within the tool.
- Generator – its task is to create the frames, based on the configuration file, using the PacketBuilder for this purpose. Generator is also sending the generated traffic into the network.

AP modules:

- Receiver – this module receives the frames on the other side of the topology, transfers the frames to the user interface for viewing and also responds to the traffic.
- Analyzer – its task is to evaluate the received frames and to check if the received traffic is in the accordance with the test scenarios.
- Response Checker – this module is checking the response based on the type of protocol, used for testing, and it provides the results to the AP.

C. *Configuration file*

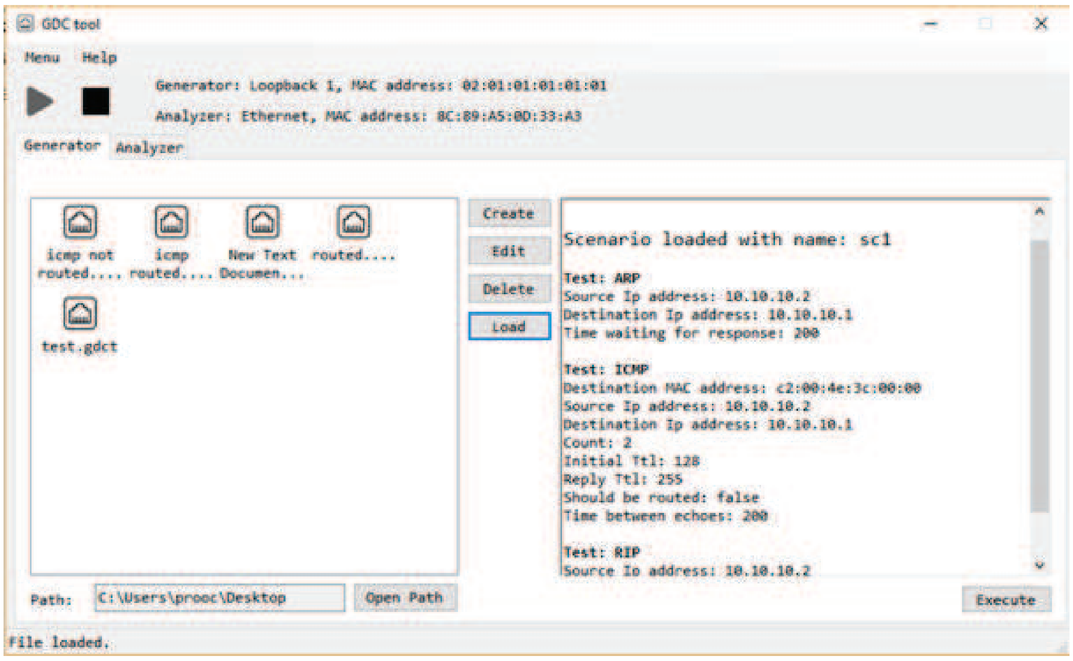
The scenarios for testing the networking hardware are represented by a configuration file (CF) loaded to the tool and then executed. The server side is building the packets stream according to the CF and the client side validates the traffic, incoming from the other side of the topology, based on the CF.

The CF is represented as an XML (eXtensible Markup Language) file with a custom suffix (e.g. gdct – GDC tool) for simple recognition by the tool. The XML form is used due to simplicity and modularity. XML files are easy to edit with various tools. A very simple example of CF is shown in Fig. 3.

The example illustrates the CF format. Each scenario has a name and contains several tests, distinguished by a description that represents the test type (i.e. targeted functionality). Inside the test hierarchical level of the XML structure, there are the actual test options. In the

```
<scenario name = "sc1">
  <test description = "arp">
    <sip>10.10.10.2</sip>
    <dip>10.10.10.1</dip>
    <wait>200</wait>
  </test>
  <test description = "ICMP">
    <dmac>c2:00:25:f0:00:00</dmac>
    <sip>10.10.10.2</sip>
    <dip>20.20.20.2</dip>
    <count>4</count>
    <inttl>128</inttl>
    <outttl>127</outttl>
    <routing>true</routing>
    <wait>200</wait>
  </test>
</scenario>
```

Figure 3. Configuration file example



```
Generator: Loopback 1, MAC address: 02:01:01:01:01:01
Analyzer: Ethernet, MAC address: 8C:89:A5:0D:33:A3
```

Path: C:\Users\prooc\Desktop

```
Scenario loaded with name: sc1

Test: ARP
Source Ip address: 10.10.10.2
Destination Ip address: 10.10.10.1
Time waiting for response: 200

Test: ICMP
Destination MAC address: c2:00:4e:3c:00:00
Source Ip address: 10.10.10.2
Destination Ip address: 10.10.10.1
Count: 2
Initial Ttl: 128
Reply Ttl: 255
Should be routed: false
Time between echoes: 200

Test: RIP
Source Ip address: 10.10.10.2
```

```
File loaded.
```

Figure 4. The GDC tool – the main window of the graphical user interface with Generator screen displayed

example of *arp* test, there are specified the source IP (Internet Protocol) address (*sip*), the destination IP address (*dip*), and how much time the tool will wait for the response (*wait*). There are even more options for the *ICMP* test.

D. *Graphical User Interface*

The usability requirement of our solution is accomplished by the use of GUI. Fig. 4 illustrates the main window of the GUI with the Generator screen displayed.

The Generator screen contains the CF control elements with the available CFs displayed on the left-hand side and the content of the loaded CF displayed on the right-hand side text field. Based on loaded CF the tests can be initiated using the Execute button. However, prior to this the devices to be tested have to be selected and switched on. The connected devices can be accessed by means of Menu, available in the upper part of the main window. The pop-up window is invoked, illustrated in Fig. 5, enabling to display and select the devices.

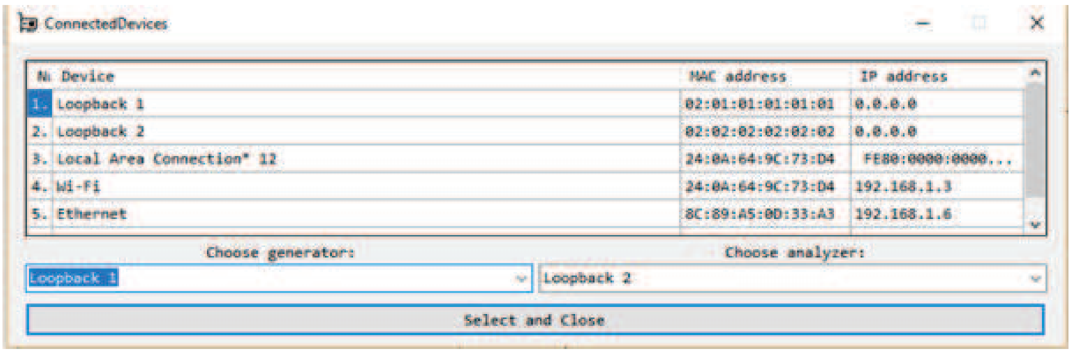
After the tests initiation the main window is automatically switched to the Analyzer screen, illustrated in Fig. 6, displaying the actual state of the initiated tests. The results of tests are also saved into a text file.

The list of filtered received packets can be tracked in the receiver window of Analyzer, illustrated in Fig. 7 (available by means of Menu). The filter area extracts the protocol type and the exact IP address to the packet list.

IV. THE SOLUTION VERIFICATION AND TESTING

As a development environment, Microsoft Visual Studio Enterprise 2015 and Visual C# 2015 were used to implement the proposed tool. The C# language was extended by Pcap.net library [8], which is taking care of communication with network interfaces and it is capable to access frames and network interfaces at the higher level. The Pcap.net library supports all the protocols needed to implement this tool except for the RIP protocol that had to be implemented manually.

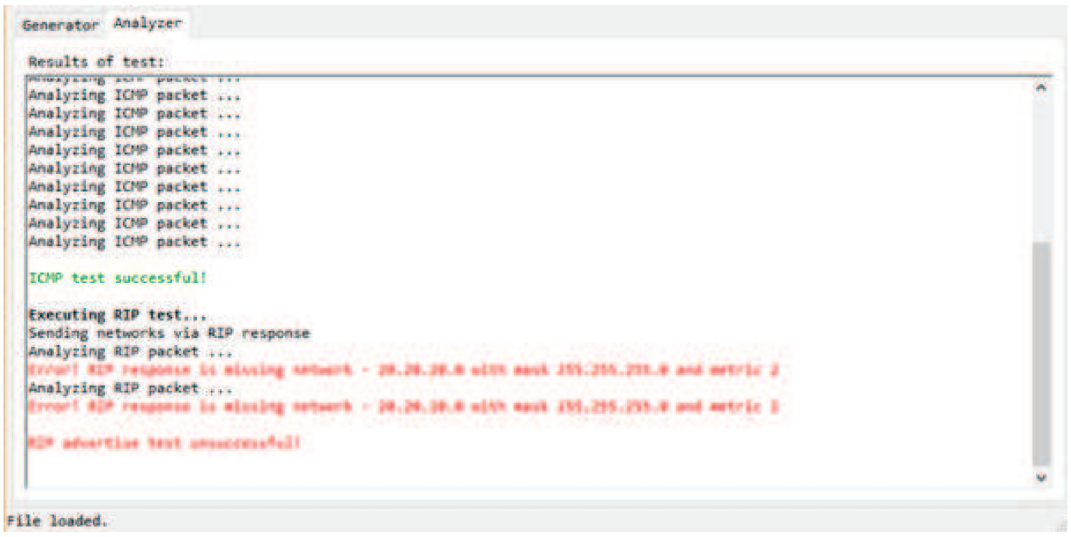
The proposed and implemented tool was verified in two phases. The first phase was the continuous testing during the design and implementation of the individual solution modules. The next phase was the overall testing of the whole solution.



№	Device	MAC address	IP address
1.	Loopback 1	02:01:01:01:01:01	0.0.0.0
2.	Loopback 2	02:02:02:02:02:02	0.0.0.0
3.	Local Area Connection* 12	24:0A:64:9C:73:D4	FE80:0000:0000...
4.	Wi-Fi	24:0A:64:9C:73:D4	192.168.1.3
5.	Ethernet	8C:89:A5:0D:33:A3	192.168.1.6

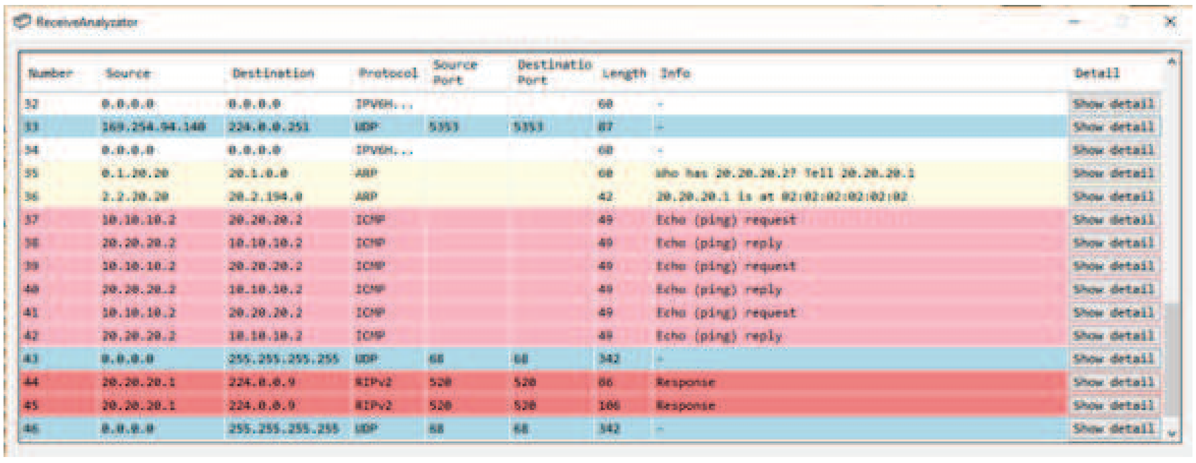
Choose generator: Loopback 1
Choose analyzer: Loopback 2

Figure 5. The GDC tool – connected devices selection



```
Results of test:
Analyzing ICMP packet ...
Analyzing ICMP packet ...
Analyzing ICMP packet ...
Analyzing ICMP packet ...
Analyzing ICMP packet ...
Analyzing ICMP packet ...
Analyzing ICMP packet ...
Analyzing ICMP packet ...
Analyzing ICMP packet ...
Analyzing ICMP packet ...

ICMP test successful!

Executing RIP test...
Sending networks via RIP response
Analyzing RIP packet ...
Error! RIP response is missing network - 20.20.20.0 with mask 255.255.255.0 and metric 2
Analyzing RIP packet ...
Error! RIP response is missing network - 20.20.20.0 with mask 255.255.255.0 and metric 1

RIP advertise test unsuccessful!

File loaded.
```

Figure 6. The GDC tool – the Analyzer screen of the main window



Number	Source	Destination	Protocol	Source Port	Destinatio Port	Length	Info	Detail
32	0.0.0.0	0.0.0.0	IPV6H...			60	-	Show detail
33	169.254.94.140	224.0.0.251	UDP	5353	5353	87	-	Show detail
34	0.0.0.0	0.0.0.0	IPV6H...			60	-	Show detail
35	0.1.20.20	20.1.0.0	ARP			60	Who has 20.20.20.2? Tell 20.20.20.1	Show detail
36	2.2.20.20	20.2.194.0	ARP			42	20.20.20.1 is at 02:02:02:02:02:02	Show detail
37	10.10.10.2	20.20.20.2	ICMP			49	Echo (ping) request	Show detail
38	20.20.20.2	10.10.10.2	ICMP			49	Echo (ping) reply	Show detail
39	10.10.10.2	20.20.20.2	ICMP			49	Echo (ping) request	Show detail
40	20.20.20.2	10.10.10.2	ICMP			49	Echo (ping) reply	Show detail
41	10.10.10.2	20.20.20.2	ICMP			49	Echo (ping) request	Show detail
42	20.20.20.2	10.10.10.2	ICMP			49	Echo (ping) reply	Show detail
43	0.0.0.0	255.255.255.255	UDP	68	68	342	-	Show detail
44	20.20.20.1	224.0.0.9	RIPv2	520	520	86	Response	Show detail
45	20.20.20.1	224.0.0.9	RIPv2	520	520	106	Response	Show detail
46	0.0.0.0	255.255.255.255	UDP	68	68	342	-	Show detail

Figure 7. The GDC tool – Analyzer receiver window

For the frames capturing and displaying, Wireshark version 2.2.5 was used, which also captures the network traffic in promiscuous mode. For test purposes, the GNS3 in version 0.8.7 was also used, in which the virtual router implementation tool was attached.

Different methods were used to verify the accuracy of the test during the continuous testing. Their form differed from module to module.

The configuration module was tested during each function implementation using the debugging mode in the development environment. For this module implementation the functions directly implemented in Visual C# libraries have been used, therefore their testing was not crucial. However, it was always checked whether the implemented feature provided the desired result. During these tests, several possible erroneous inputs, which could have occurred, were always been treated.

The generator was tested using Wireshark. Firstly, a frame was generated according to the requirements, and then, it was compared to the captured frames on the interface. The frame fields' validity was also checked. Thus, the functionality of the frame creation functions was also tested.

When testing the receiver, again the Wireshark was used, this time to compare the received and displayed frames in both tools. The receiver was tested using the already implemented generator.

For verification of the implemented analyzer, the already verified generator and receiver modules were used. The input argument for this test was the configuration file from which the generator generated and sent the frames, the receiver received them, and the analyzer output was the information displayed in the GUI. This result was checked based on the changes in the CF and changes in the test topology.

The overall solution testing was performed when all the modules were already implemented and verified separately. Firstly, the overall solution was tested using GNS3 with a router attached to each network interface. The router was connected using the Microsoft Loopback virtual network interfaces. The tests were performed using fully functional router, as well as the software router implemented by a student. During the overall solution testing the compliance with the specified requirements was also verified.

V. Conclusions and Further Work

In this paper, we have proposed a tool for generation of a diagnostic communication. The analyzed existing tools are primarily focused on general diagnostics of a network traffic (e.g. as a prevention of attacks) and it is too difficult to use them to test a specific functionality of a network element. The proposed tool can generate a network traffic based on the specified test scenario in the configuration file, it captures responses of the devices in the tested network, and evaluates the test. This way, the proposed tool can be used to verify the functional aspects of the connected network devices in the tested network.

Although it was mainly intended for students to verify their implementations of software routers (i.e. a coursework assignment during the university study) during development, it can be also used by teachers to quickly evaluate the correctness of the final implemented solutions. Moreover, it can be used to test the functionality of real hardware network devices.

The implemented prototype of the proposed tool was verified on the selected testing scenarios. These tests showed that the tool is useful and it speeds up the checking of correct functionality of the tested network device. The tool has its limitations regarding general usage (e.g. support of the limited number of network protocols); however, due to its modular architecture, it can be easily extended and the limitations eliminated.

Acknowledgment

This work was mainly supported by the Slovak Cultural and Educational Grant Agency (KEGA 011STU-4/2017) 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] S. Yegulalp, “Review: 6 slick open source routers,” InfoWorld, 2012. [Online; accessed September 2017]. Available: http://www.infoworld.com/article/2615872/networking/networking-review-6-slick-open-source-routers.html.

[2] S. Floyd, „Traffic Generators for Internet Traffic,“ National Science Foundation, September 2010. [Online; accessed October 2017]. Available: http://www.icir.org/models/trafficgenerators.html.

[3] J. Sommers, “Harpoon – A flow-level traffic generator,” 2005. [Online; accessed September 2017]. Available: http://cs.colgate.edu/~jsommers/harpoon/.

[4] P. Srivats, “OSTINATO: Network Traffic Generator and Analyzer,” 2017. [Online; accessed September 2017]. Available: http://ostinato.org.

[5] Northwest Performance Software, Inc., “NetScanTools Pro: Packet Generator Tool,” 2017. [Online; accessed September 2017]. Available: http://www.netscantools.com/nstpro_packet_generator.html.

[6] 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.

[7] U. Lamping, R. Sharpe, and E. Warnicke, “Wireshark User’s Guide: For Wireshark 2.1”, 2014. [Online; accessed September 2017]. Available: https://www.wireshark.org/download/docs/user-guide-us.pdf.

[8] B. Brickner, „Pcap.Net,“ GitHub, 2017. [Online; accessed October 2017]. Available: https://github.com/PcapDotNet/Pcap.Net.