ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 8, No. 1, 2023

THE NETWORK LOAD BALANCER IN DECENTRALIZED SYSTEMS

Nazarii Klymyshyn

Lviv Polytechnic National University, 12, S. Bandery str., Lviv, 79013, Ukraine
Author's e-mail: klymyshyn.nazarii.mkisp.2022@lpnu.ua

https://doi.org/10.23939/acps2023.01.025

Submitted on 23.02.2023

***Abstract*: This article explores the implementation of network load balancing in decentralized systems using OpenWrt, Quality of Service (QoS), and traffic balancing techniques. The increasing demand for high-quality network services and the surge in network traffic requires the adoption of more efficient load-balancing methods to maintain network performance. This paper discusses the use of OpenWrt, an open-source firmware for network routers, to configure and manage network traffic. The article also covers the implementation of QoS and traffic balancing techniques to optimize network performance and reduce network congestion. The study employs iperf3 to evaluate network performance and demonstrates the effectiveness of the proposed network load-balancing approach. The index terms include OpenWrt, QoS, balancing, traffic, and iperf3.**

***Index Terms*: OpenWrt; QoS; balancing; traffic; iperf3.**

I. INTRODUCTION

Over the last decade, the needs and requirements of Internet users for a reliable connection channel have increased significantly, which would allow balancing network traffic between several channels and maintainning the Internet connection in case of failure of one or more channels. Such requirements are explained by the need to deploy various services or remote work in places where there is no professional network infrastructure. Deploying a secure Internet connectivity infrastructure should be simple and accessible to the lay user [1].

The best way to meet user needs is to develop a network traffic load balancing package for the OpenWrt operating system, which is supported by many modern network router manufacturers [2].

Most research on network traffic load balancing has focused on the problem of software-defined networking (SDN). Software-configured networks have a more complex architecture and more network devices compared to an end-user network, so most users cannot - afford it [3].

Sometimes computer networks fail, making the Internet or certain network segments unavailable from the current node. As a rule, failures occur within a specific provider, which leads to connection problems only for a certain group of users served by that provider.

The computing power of computer networks must increase as the amount of data transferred over the network and the number of nodes on the network increase. This goal can be achieved by replacing and improving the hardware components of the network, but this method is expensive and requires additional resources. The load balancing mechanism on the existing network node reduces the response time and allows for the even use of resources [4].

In Fig. 1 we see network traffic balancing architecture in cloud computing.

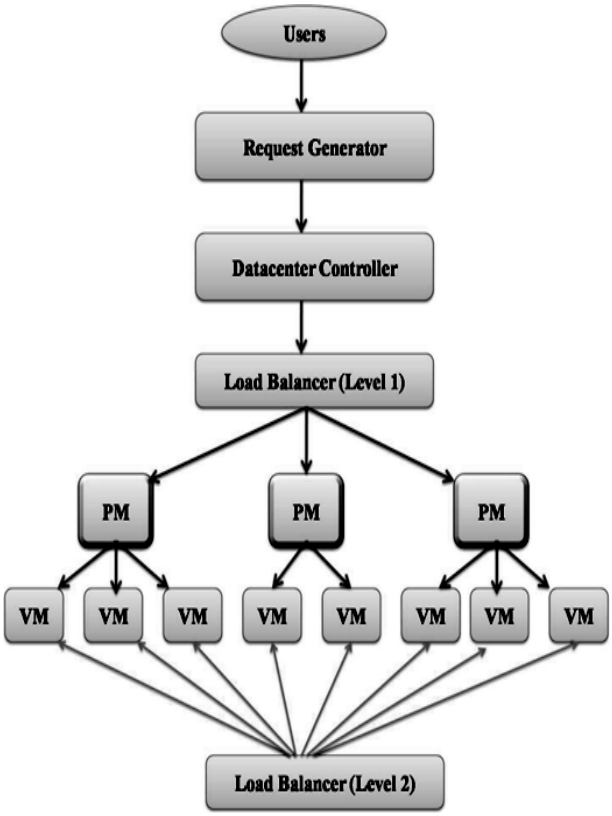

Fig. 1. Network traffic balancing architecture in cloud computing

II. TRAFFIC BALANCING PURPOSES

The computing power of computer networks must increase following the growth of the amount of data transmitted over the network and the number of nodes in the network. This goal can be achieved by replacing and improving network hardware components, but this method is expensive and a waste of resources. Load

balancing on an existing network node reduces response time and allows for even resource utilization.

1) Goals of traffic balancing

The goals of traffic balancing are [5]:

- Improvement of QoS service in the network, and therefore the increase of efficiency and productivity of the system. The goal of QoS is to provide users with improved service by preventing excessive delays, reducing response times, and optimizing performance.
- Optimizing the use of resources. Correct balancing results in an even and economical use of resources: bandwidth, memory usage, and processor time.
- Reduction of data transmission delays. Data transmission delays are the time it takes for a router to transmit a unit of data. Delays depend on the performance of the node, the fullness of the data transmission queue, and the size of the packet.
- Reduction of response time. Response time is the time between receiving a request and sending a response.
- Avoidance of narrow areas. To avoid bottlenecks during data transmission, it is recommended to evenly distribute the load on available channels.
- Increasing bandwidth. Throughput is the amount of correctly sent data over a certain period from the source to the destination.

In Fig. 2 and Fig. 3 we see the difference between cross-zone load balancing enabled and disabled states.

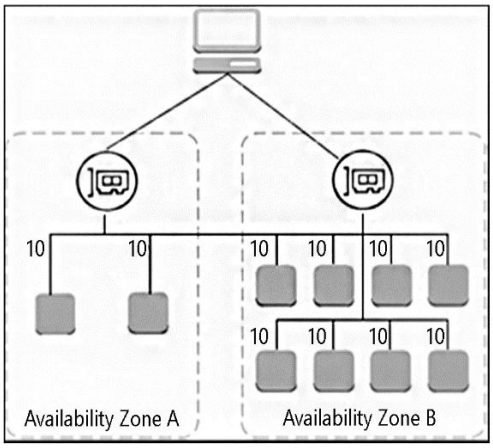

Fig. 2. Network traffic when cross-zone load balancing is enabled

2) Traditional methods of balancing in SDN

A large number of different balancing algorithms are used in SDN, each of which is designed to solve certain problems and has its advantages and disadvantages [6].

For example, in networks with intensive traffic, the "Middlebox" algorithm is used, which helps reduce data transmission delays and optimizes the use of bandwidth. The essence of the algorithm is to use an additional controller, which is a Clos network and is responsible for the uniform distribution of traffic between other network segments while supporting QoS technology. The disadvantage of this approach is the failure of the entire network in case of failure of the controller.

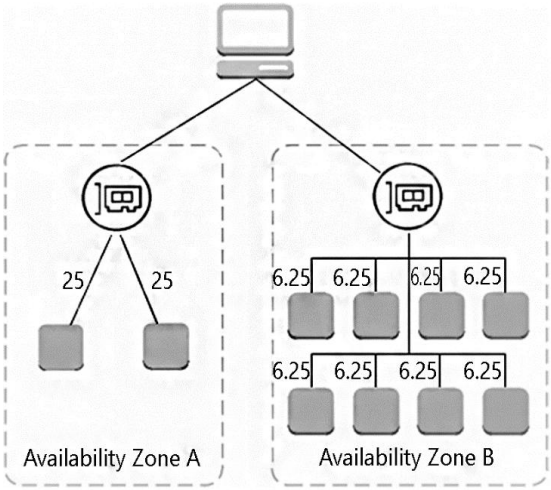

Fig. 3. Network traffic when cross-zone load balancing is disabled

In addition, in SDN networks, a topology with several balancing controllers can be used. The multi-controller deployment algorithm balances the load between several regionally distributed controllers. The number of controllers for each region in this case depends on the number of network equipment in specific regions. This algorithm offers reduced data transmission delays and efficient balancing, but its disadvantages are poor network security and limited use of QoS technology.

Some algorithms use the migration of network equipment from more loaded controllers to less loaded ones, which is a rather time-consuming operation, but often justified in conditions of significant asymmetric network load. The "Flow Stealer" algorithm is a simple balancing algorithm that listens for events that are transmitted between different controllers and allows you to temporarily take parts of data flows from more loaded controllers to less loaded ones. This method responds quickly to network events and minimizes the number of network equipment migrations.

Some algorithms are based on the active use of QoS metrics, which allows for dynamic balancing, optimizing resources, and searching for the shortest routes. Such algorithms include modifications of the dynamic balancing algorithm.

Algorithms also use the idea of actively changing traffic routes when there is a change in the network. A novel load-balancing algorithm applies this idea while increasing the bandwidth and reducing the number of lost packets. This algorithm suggests rerouting if the bandwidth is less than a certain threshold value. The disadvantage of this approach is that frequent rerouting can be a complex and/or time-consuming operation.

Cross-zone load balancing is a technique used in distributed computing systems to balance the load across

multiple availability zones in a cloud infrastructure. An availability zone is a separate physical location within a cloud region that is designed to be isolated from failures in other zones.

When a load balancer distributes traffic across instances in multiple availability zones, it can be configured to either distribute the traffic evenly across all zones or prioritize one zone over others. In the case of cross-zone load balancing, the traffic is distributed evenly across all availability zones.

This approach helps to ensure the high availability and fault tolerance of an application by distributing traffic across multiple zones. Traffic can be automatically redirected to the healthy instances in other zones if one zone experiences a failure. Cross-zone load balancing can also improve the overall performance of an application by reducing the latency of requests and improving the responsiveness of the system.

Some cloud service providers like Amazon Web Services (AWS) support cross-zone load balancing as the default behavior for their load-balancing services. It is important to note that while cross-zone load balancing can help to improve the resilience and performance of an application, it may also increase the cost of running the application by requiring more resources to maintain the availability of the application across multiple zones.

III. REVIEW OF RELATED WORKS

The article [7] discusses the use of reinforcement learning (RL) techniques to optimize load balancing in data center networks. Specifically, the authors propose a new RL-based load-balancing algorithm that takes into account the current network traffic load, network topology, and available network resources to make intelligent decisions about how to distribute network traffic across the data center.

The proposed algorithm is evaluated through simulations and experiments on a testbed, and the results show that it outperforms traditional load-balancing algorithms in terms of network throughput, network utilization, and response time.

Overall, the article presents an interesting approach to using RL techniques to optimize load balancing in data center networks, which are becoming increasingly complex and challenging to manage. The use of RL techniques has the potential to improve network performance and efficiency and may be of interest to professionals working in network engineering and data center management.

In this paper [8] authors put forth a mechanism for load-balancing routing in mesh wireless networks based on SDN that takes into account QoT constraints. The mechanism dynamically distributes traffic among network paths to optimize network performance while ensuring QoT requirements are met.

The authors demonstrate the effectiveness of the proposed mechanism by comparing it with existing routing algorithms. Results show improvements in network throughput, packet delivery ratio, and end-to-end delay, with the mechanism meeting QoT requirements.

The article discusses practical applications of the proposed mechanism, such as in smart city networks and IoT applications. It offers a promising solution to enhance network performance while considering QoT constraints in modern communication systems.

In summary, the article presents a novel approach to load-balancing routing in mesh wireless networks that consider QoT constraints, with simulation results demonstrating improved performance compared to existing algorithms. The proposed mechanism has practical applications in various fields and offers potential benefits for network performance, flexibility, and scalability.

The article [9] discusses the use of software-defined networking (SDN) technology to improve load balancing on campus networks (CNs). CNs are local area networks (LANs) that are typically used in educational institutions, government organizations, and corporate campuses.

The authors propose a new load-balancing architecture that utilizes SDN technology to dynamically allocate network resources based on the current network traffic load. The proposed architecture consists of three layers: the user access layer, the network aggregation layer, and the core layer, each of which is responsible for different aspects of load balancing.

The proposed architecture is evaluated through testing, and the results show that it outperforms other load-balancing techniques in terms of network throughput, delay, and packet loss rate, particularly under high traffic loads.

The article presents an interesting approach to improving load balancing on CNs using SDN technology, which can help to optimize network performance and improve the user experience. Professionals working in network engineering and IT infrastructure, particularly those responsible for managing campus networks may pay attention to this paper.

The article [10] discusses the use of software-defined networking (SDN) technology to implement dynamic load balancing in data center networks (DCNs). DCNs are critical for delivering high-performance computing and storage services, and load balancing is an important technique for optimizing the use of network resources and ensuring network reliability.

The authors propose a new dynamic load-balancing algorithm that utilizes SDN technology to monitor network traffic and allocate network resources in real time. The proposed algorithm considers factors such as network topology, available bandwidth, and network congestion levels to make intelligent decisions about how to distribute network traffic across the DCN.

The proposed algorithm is evaluated through simulations, and the results show that it outperforms traditional load-balancing techniques in terms of network throughput, network utilization, and response time, particularly under high traffic loads.

The development of a software system [11] for motion detection and tracking. The system is designed to identify and track different types of motion and objects in real-time video streams. The work proposes advanced algorithms for testing and processing investigation results, which allows it to differentiate between various types of motion and objects.

IV. OBJECTIVES

The main objective of this article is to present a new network load-balancing algorithm that can outperform existing algorithms. The algorithm is aimed at achieving better network performance, by distributing traffic more efficiently across multiple servers in a network.

To achieve this objective, we will firstly conduct a thorough review of existing network load-balancing algorithms and identify their limitations. Based on this analysis, we will propose a new algorithm that can address these limitations and provide better performance.

We will then compare the performance of the new algorithm with existing algorithms using a series of tests. These tests will be conducted under various traffic loads, network topologies, and configurations to ensure the robustness and reliability of the new algorithm.

To verify the effectiveness of the new algorithm, we will evaluate the performance of the network load balancer by measuring various network parameters such as average values, maximum values, minimum values, and mean square deviations. Bandwidth should increase by at least 30 percent. This metric will help us to gain a comprehensive understanding of the performance of the new algorithm and its ability to handle different types of network traffic.

Overall, the objectives of this article are to present a new and innovative network load-balancing algorithm that can provide superior network performance and to verify its effectiveness through rigorous testing and evaluation.

V. REQUIREMENTS TO PERFORM MEASUREMENTS

1) Requirements for the experiment environment

To ensure the high-quality conduct of the experiment, it is necessary to define a list of requirements for the environment in which the experiment will be conducted, which relate to the next topics.

- The device on which balancing will be carried out;
- Involved communication channels;
- Connected devices of end users participating in data transmission;
- Additional bandwidth measuring device on the balancing device.

2) Requirements for the device on which network traffic is balanced

The routing device can be any physical device that supports the OpenWrt version 21.02 operating system. This category includes both devices directly intended for routing network traffic and general-purpose devices such as Raspberry Pi:

- The use of a virtual machine (for example QEMU, VirtualBox, etc.) is not allowed, as it may distort the test results.
- A device must have more than one physical port to establish a connection. It is acceptable to use both Ethernet ports and USB ports in any combination.
- The device must have the physical ability to work as a Wi-Fi access point or have sufficient physical ports available to connect all clients and connection channels.

3) Requirements for physical communication channels

To conduct experiments, it is necessary to use at least two physical communication channels with the same characteristics (the same bandwidth and delays). The need for symmetry of communication channels during the experiment exists due to the possibility of different characteristics distorting the results of the experiment:

- It is allowed to use any type of physical communication channel if they have the same characteristics.
- It is recommended to use two physical communication channels of the same type for experiments. A combination of Ethernet cables and USB hubs is allowed if the bandwidth of the USB hub is not lower than the bandwidth of the Ethernet cable.
- USB hubs must be compatible with the OpenWrt operating system version 21.02.
- The total bandwidth of the Wi-Fi network must be higher than the bandwidth of each of the physical channels of the connection.

4) Requirements for end-user devices

- It is necessary to use at least one end-user device for experiments, at least two are recommended:
- Ability to connect via Wi-Fi or through a physical connection if permitted by the experiment.
- Ability to use the preferred bandwidth measurement tool on the end user's device. It is recommended to use the iperf3 tool.
- It is necessary to make sure that during the experiment the device does not carry out significant operations with the network (uploading or downloading a large amount of data not related to the experiment) to avoid distorted results.

5) Requirements for an additional bandwidth measuring device on the balancing device

- The device can be a routing device or a general-purpose device.
- The device must support the same number of physical communication channels that will be tested under experimental conditions.
- The user should be able to use the desired bandwidth measurement tool.
- The bandwidth of the physical communication channels should not be lower than the bandwidth of the

physical communication channels of the balancing device, or it is necessary to make sure in advance that the device is capable of supporting the bandwidth corresponding to the experimental conditions.

- The processor specifications of the device must not be lower than the processor specifications of the balancing device, or it is necessary to ensure in advance that the device is capable of supporting the bandwidth corresponding to the experimental conditions.

VI. DESIGNING THE ARCHITECTURE OF THE COMPONENT

When designing the software component, the modular architecture of the OpenWrt operating system was analyzed and it was determined that the software component for load balancing should be designed as an independent module of the operating system, which is contained in the user space and defines dependencies on other independent modules of the system:

Netfilter, iptables, ip route, teach, network manager.

UML notation was used to design the diagrams of the software component, which allows you to construct diagrams to display the structure, behavior, and object component of the system. A UML use case diagram is provided in Fig. 4.

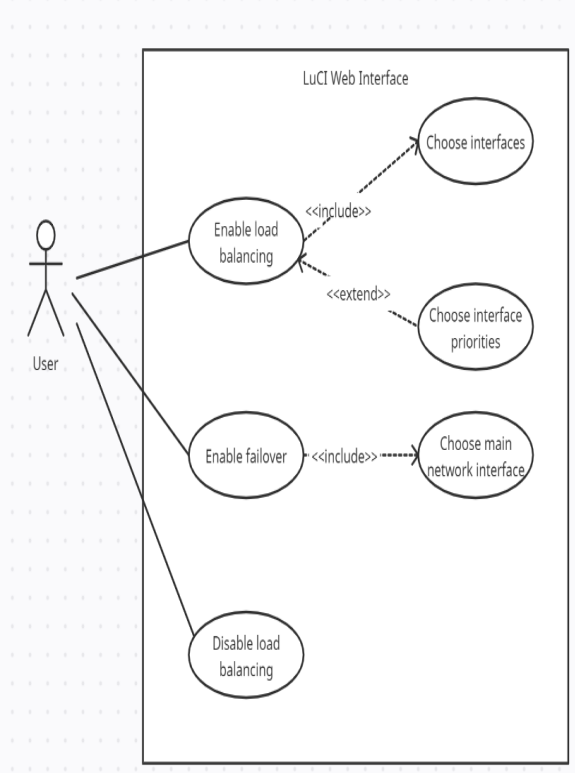

Fig. 4. Use case diagram

1) Use case diagram

To describe the use cases of the software component, it was chosen to define possible user actions using the UML language, namely to build a UML case diagram.

- The user can interact with the presented system by changing the configuration file and enabling or disabling the software component using the command line.
- Users can enable the software component for balancing using the command line, while it is necessary to specify the logical network interfaces on which network traffic will be balanced, as well as specify the priorities of the selected interfaces using the configuration file;
- Users can enable the failover software component using the command line, while the user must specify the primary and backup logical network interfaces using a configuration file;
- Users can disable the component for balancing and failover.

2) Deployment diagram

The UML deployment diagram is designed to understand the membership and relationship between all the involved modules (packages) of the system. A UML deployment diagram is provided in Fig. 5.

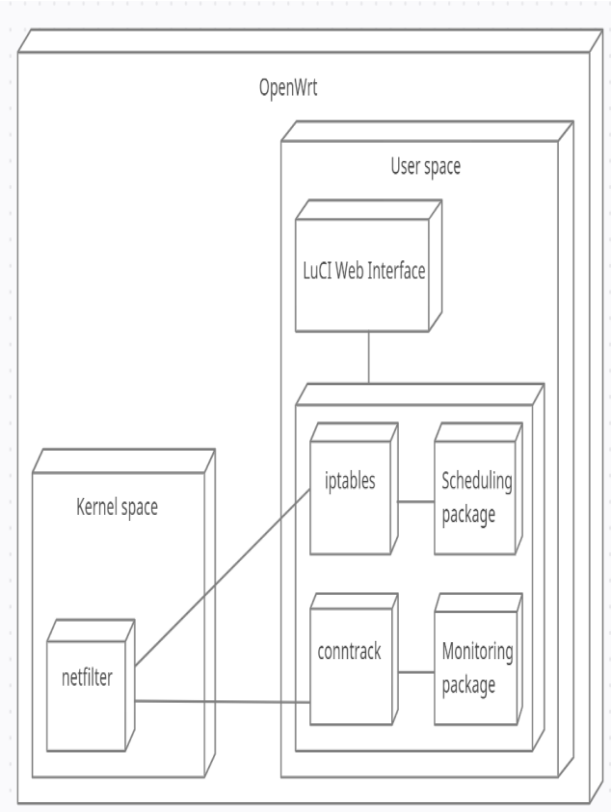

Fig. 5. Deployment diagram

The following components are located in the user space of the OpenWrt operating system:

- LuCI package for the interaction of the software component being designed with the user graphical interface.
- Package for balancing network traffic, which contains two lower-level modules and also interacts with the standard conntrack and iptables packages.

- Monitoring module – used to monitor logical network interfaces, their activity, and network interface events.
- Planning module – used to create specific rules based on received information about active network interfaces and configuration (interfaces to be used, their priorities, and metrics) specified by the user.

The kernel space contains the standard netfilter module, which acts as a firewall in the OpenWrt operating system and is used by the iptables and conntrack packages from the user space.

3) Sequence diagram

A UML sequence diagram is built to capture the sequence of actions a system takes to reach a desired state and the interactions between system components. A sequence diagram is presented in Fig. 6.

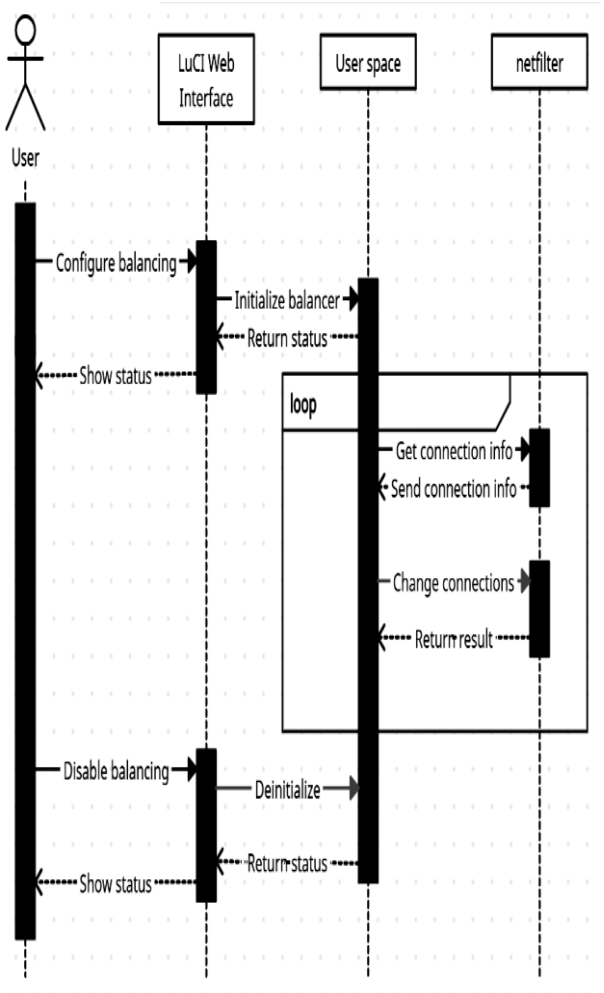

Fig. 6. Sequence diagram

The diagram shows the following system modules:

- LuCI – a user-space package for user interaction with a network traffic balancing package;
- user space – all packages participating in network traffic balancing and belonging to the user space (except LuCI);
- netfilter module – a firewall contained in the kernel space.

The sequence of system actions is as follows:

- the user initiates the activation of the software component for balancing using the LuCi package;
- LuCI package sends the initialization request to the user space, namely to the load balancing package;
- load balancing package returns the current status of the LuCi package and starts its work;
- LuCI package displays the status to the user;
- load balancing package sends a request about current connections to netfilter;
- netfilter returns the result of the request to the load-balancing component;
- load balancing package processes the received results and, if necessary, sends a request to change firewall rules or routes to the netfilter module;
- the netfilter module executes a request and returns the result to the load-balancing package;
- the last four steps are repeated at intervals as long as the network load balancing component remains enabled;
- the user sends a request to disable the network load balancing component using the LuCI graphical user interface;
- the LuCi package sends a request to the network traffic balancing component;
- the network traffic balancing package releases captured resources and performs deinitialization, and returns the result;
- LuCI package displays the status to the user.

VII. DETAILS OF THE SOFTWARE IMPLEMENTATION

The source of information about updating the state of the logical network interface is the network manager component of the OpenWrt operating system.

To simplify interaction with the network manager component, the “hotplugd” service was used, as it is convenient to use in bash scripts: “hotplugd” allows you to “subscribe” to an event and call a certain command or script when this event occurs.

The load balancing component primarily works with the iptables mangle table. This table allows you to modify and track packet headers for various purposes.

In this case, packets belonging to a particular connection are marked with a mangle table so that they can be assigned to a particular network interface, and responses to these packets are not confused and sent correctly to the recipient.

For load balancing to work correctly, you need to create a sufficient number of rules: some of them will apply to the specific configuration of the user, and some of them will be general rules that ensure smooth routing when applying a non-trivial configuration of physical connections.

Each of the balancing mechanisms selected in section 2 must have its own rule or set of rules.

VIII. EXPERIMENTAL RESULTS AND ANALYSIS

1) Research environment

Following the conditions for the environment of the experiment, the proper devices for the study were configured.

The device under test is a device on the Raspberry Pi 3B+ platform that supports the OpenWrt operating system (18, 19) and a software component for network load balancing.

iperf3 client – two devices connected to the device under test using a Wi-Fi network. The devices include the iperf3 package to act as a client during testing.

iperf3 server – is a device connected to the device under test using two Ethernet cables, one of which is connected to the device under test using a USB adapter.

Connections from the tested device to the iperf3 server are symmetrical, each with a maximum bandwidth of 72 Mbit/s. A schematic representation of the described setting is shown in Fig. 7.

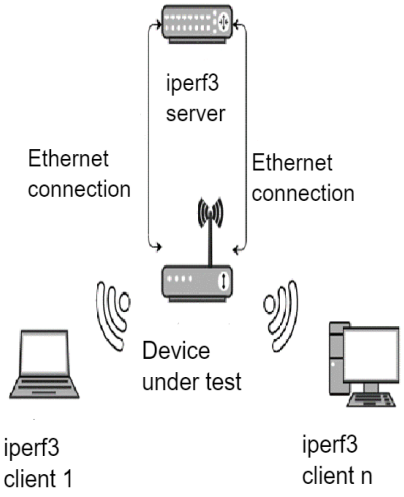

Fig. 7. Research environment

2) Obtained results

During the research, each of the developed balancing mechanisms was tested. Each engine was tested with iperf3 15 times at 30-second intervals. Given that the given tool displays the results every second, the total number of point results is 450 for each considered mechanism.

For demonstration, the most important metrics of each iteration were collected: the average, minimum, and maximum throughput of the sender and the number of retries when sending data. The results of testing the balancing mechanism based on the selection of multiple packages are shown in Fig. 8.

The results of testing the mechanism for limiting the number of connections per unit of time are plotted in Fig. 9.

The results of testing the balancing mechanism using the probability of entering the channel are provided in graphs in Fig. 10.

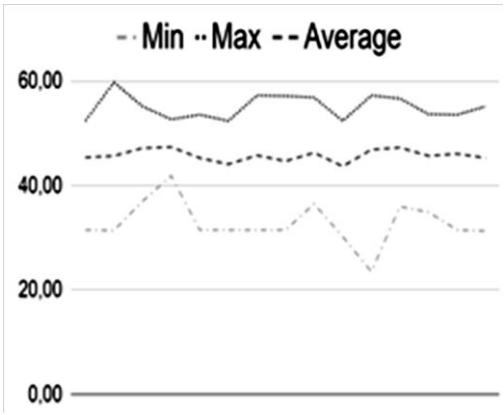

Fig. 8. Test results of the balancing mechanism based on the selection of multiple packages, device #1

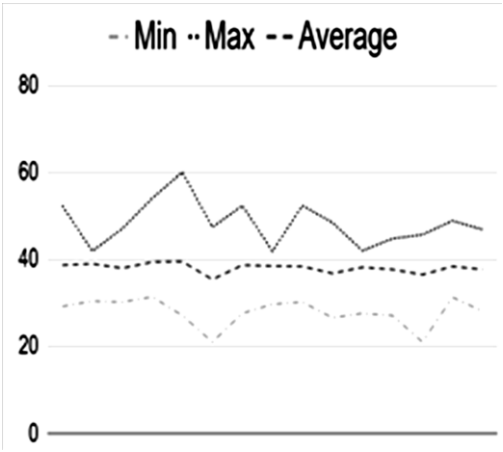

Fig. 9. Test results of the mechanism for limiting the number of connections per unit of time, device No. 1

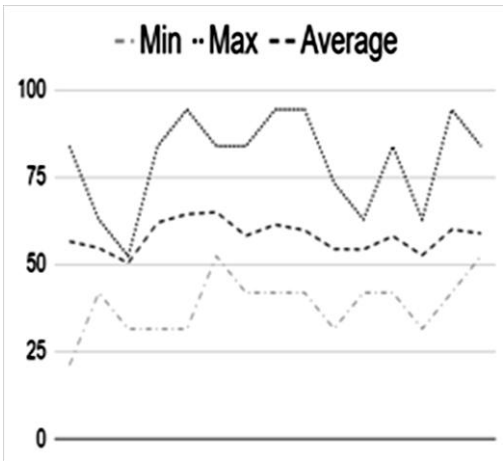

Fig. 10. The results of testing the balancing mechanism using the probability of getting into the connection channel, device No. 1

3) Evaluation of the reliability of the obtained results

To properly assess the reliability of the obtained results, it was decided to calculate the following values: average va-

lue, maximum value, minimum value, mean square deviation, and the representativeness error of the arithmetic mean.

Table 1 shows the results of testing the first mechanism based on multiple package selections on device No. 1.

Table 1

Results of testing the balancing mechanism based on multiple package selection on device No. 1

No. (i)	$Min_{1,i}$, mbit/s	$Max_{1,i}$, mbit/s	$Av_{1,i}$, mbit/s	retries amount
1	31.5	52.4	45.4	0
2	31.4	59.8	45.7	0
3	37.0	55.2	47.2	0
4	41.9	52.7	47.4	0
5	31.5	53.6	45.3	0
6	31.5	52.4	44.1	0
7	31.5	57.3	45.8	0
8	31.5	57.2	44.7	0
9	36.5	56.9	46.3	0
10	30.2	52.4	43.7	0
11	23.5	57.3	46.9	0
12	36.0	56.7	47.3	0
13	34.9	53.7	45.7	0
14	31.5	53.6	46.1	0
15	31.3	55.2	45.3	0

Table 2 presents the results of testing the second mechanism for limiting the number of connections per unit of time on device No. 1.

Table 2

The results of testing the mechanism for limiting the number of connections per unit of time on device No. 1

No. (i)	$Min_{2,i}$, mbit/s	$Max_{2,i}$, mbit/s	$Av_{2,i}$, mbit/s	retries amount
1	29.2	52.4	38.7	0
2	30.4	41.9	39.0	0
3	30.2	47.1	38.0	0
4	31.4	54.1	39.4	0
5	27.1	60.1	39.5	0
6	21.0	47.4	35.4	0
7	27.6	52.4	38.7	0
8	29.7	41.9	38.5	0
9	30.2	52.4	38.4	0
10	26.6	48.5	36.8	0
11	27.6	42.0	38.2	0
12	27.1	44.8	37.7	0
13	21.0	45.7	36.5	0
14	31.3	48.9	38.4	0
15	28.1	46.9	37.7	0

Table 3 presents the results of testing the third balancing mechanism using the probability of getting into the connection channel on device No. 1.

Measurements used for further calculations are provided in the Tables. Here are only provided results of testing from device No. 1.

4) Arithmetic mean value

Arithmetic averages are calculated with the formula:

$$Av_y = \frac{\sum_{i=1}^{n} Av_{y,i}}{n}. \quad (1)$$

Table 3

The results of testing the balancing mechanism using the probability of getting into the connection channel on device No. 1

No. (i)	$Min_{3,i}$, mbit/s	$Max_{3,i}$, mbit/s	$Av_{3,i}$, mbit/s	retries amount
1	21.0	83.9	56.6	0
2	41.9	62.9	54.7	0
3	31.5	52.4	50.4	0
4	31.5	83.9	62.0	0
5	31.5	94.4	64.4	0
6	52.4	83.9	65.0	0
7	41.9	83.9	58.2	0
8	41.9	94.4	61.4	0
9	41.9	94.4	59.8	0
10	31.5	73.4	54.4	0
11	41.9	62.9	54.3	0
12	41.9	83.9	58.1	0
13	31.5	62.9	52.6	0
14	41.9	94.4	60.0	0
15	52.4	83.9	58.9	0

Using (1), where n is the number of tests (n=15 in our case), i is the sequence number of the particular iteration of the test, y is the sequence number of the tested mechanism, $Av_{y,i}$ is the average bandwidth from the i iteration of y test.

The results of testing the balancing mechanism based on the selection of multiple packages:

$$Av_1 = \frac{\sum_{i=1}^{n} Av_{1,i}}{n} = 45.8\ Mbit/s.$$

Using the same formula for the results of the mechanism of limiting connections per unit of time:

$$Av_2 = \frac{\sum_{i=1}^{n} Av_{2,i}}{n} = 38\ Mbit/s.$$

Using the same formula for the results of the probability of getting into the connection channel:

$$Av_3 = \frac{\sum_{i=1}^{n} Av_{3,i}}{n} = 58\ Mbit/s.$$

5) Maximum value

The maximum value can be retrieved for each mechanism from the respective table.

For the mechanism based on the selection of multiple packages: $V_{max_1} = Max_{1,2} = 49.8$ Mbit/s.

For the mechanism of limiting connections per unit of time: $V_{max_2} = Max_{2,5} = 59.8$ Mbit/s.

For the mechanism of the probability of getting into the connection channel: $V_{max_3} = Max_{3,5} = 94.4$ Mbit/s.

6) Minimum value

In the same way, the minimum value can be retrieved for each mechanism from the respective table.

For the balancing mechanism based on the selection of multiple packages:

$$V_{min_1} = Min_{1,6} = 23.5 \text{ Mbit/s}.$$

For the mechanism of limiting connections per unit of time:

$$V_{min_2} = Min_{2,11} = 21 \text{ Mbit/s}.$$

For the mechanism using the probability of getting into the connection channel:

$$V_{min_3} = Min_{3,1} = 21 \text{ Mbit/s}.$$

7) Mean square deviation

The mean square deviations (SD) are calculated according to the formula:

$$\sigma_y = \sqrt{\frac{\sum_{i=1}^{n} Av_{y,i} - Av_y}{n}}. \quad (2)$$

Using (2), where *n* is the number of tests (*n*=15 in our case), *i* is the sequence number of the particular test, *y* is the sequence number of the tested mechanism, $Av_{y,i}$ is the average bandwidth from the *i* iteration of *y* test, and Av_y is the arithmetic mean of $Av_{y,i}$.

The results of testing the balancing mechanism based on the selection of multiple packages:

$$\sigma_1 = \sqrt{\frac{\sum_{i=1}^{n} Av_{1,i} - Av_1}{n}} = 1.2\ Mbit/s.$$

Using the same formula for the results of the mechanism of limiting connections per unit of time:

$$\sigma_2 = \sqrt{\frac{\sum_{i=1}^{n} Av_{2,i} - Av_2}{n}} = 1.11\ Mbit/s.$$

Using the same formula for the results of the probability of getting into the connection channel:

$$\sigma_3 = \sqrt{\frac{\sum_{i=1}^{n} Av_{3,i} - Av_3}{n}} = 4.24\ Mbit/s.$$

8) Arithmetic mean representativeness error

The representativeness errors of the arithmetic mean are calculated according to the formula:

$$\Delta_y = \frac{\sigma_y}{\sqrt{n}}. \quad (3)$$

Using (3), where *y* is a sequence number of the tested mechanism, σ_y is the mean square deviation of the *y* mechanism (calculated in the previous topic) and *n* is the number of tests (*n*=15 in our case). The results of testing the balancing mechanism based on the selection of multiple packages:

$$\Delta_1 = \frac{\sigma_1}{\sqrt{n}} = 0.31\ Mbit/s.$$

The results of the mechanism of limiting connections per unit of time:

$$\Delta_2 = \frac{\sigma_2}{\sqrt{n}} = 0.27\ Mbit/s.$$

The results of the mechanism using the probability of getting into the connection channel:

$$\Delta_3 = \frac{\sigma_3}{\sqrt{n}} = 1.1\ Mbit/s.$$

9) Final results

Table 4 presents the final results.

The resulting bandwidth (in percentage terms) is calculated with the next formula:

$$Bandw_{res,y} = \frac{Bandw_{av,y,1} + Bandw_{av,y,2}}{72} * 100\%. \quad (5)$$

Using (5), where 72 is the original bandwidth, y is a sequence number of tested mechanism, $Bandw_{av,y,1}$ is the average bandwidth of device No. 1 with mechanism *y*, $Bandw_{av,y,2}$ is the average bandwidth of device No. 2 with mechanism *y*.

Resulting bandwidth of the first mechanism:

$$Bandw_{res,1} = \frac{Bandw_{av,1,1} + Bandw_{av,1,2}}{72} * 100\% = 107.6\%.$$

Table 4

Final results.

y	Average bandwidth of 1st device, $Bandw_{av,y,1}$, Mbit/s	Average Bandwidth of 2nd device, vice, $Bandw_{av,y,2}$, Mbit/s	Total Bandwidth, $Bandw_{av,y,1} + Bandw_{av,y,2}$, Mbit/s
1	45.8	31.7	77.5
2	38	28.8	66.8
3	58	44.6	102.6

This shows that growth is equal to 7.6 % (initial bandwidth is equal to 100 %, the new one is 107.6 %, so it is bigger at 7.6 %).

Resulting bandwidth of the second mechanism:

$$Bandw_{res,2} = \frac{Bandw_{av,2,1} + Bandw_{av,2,2}}{72} * 100\% = 92.7\%.$$

This shows that this mechanism provided no growth.

Resulting bandwidth of the third mechanism:

$$Bandw_{res,3} = \frac{Bandw_{av,3,1} + Bandw_{av,3,2}}{72} * 100\% = 142.5\%.$$

This shows that growth is equal to 42.5 % (initial bandwidth is equal to 100 %, the new one is 142.5 %, so it is bigger at 42.5 %). This algorithm also outperforms results (40 % bandwidth growth) from the analog technology proposed by other authors [6].

IX. CONCLUSION

The mechanism, based on the selection of multiple packages grew the total bandwidth by 7.6 % compared to using a single connection channel, which is an improvement in throughput but not significant.

The mechanism of limiting the number of connections is the only one of the considered methods that demonstrated lower bandwidth measurement results than when using a single communication channel.

The balancing mechanism using the probability of hitting the connection channel showed the best results in the study – the total bandwidth has grown by 42.5 %. This algorithm also showed better results (40 % bandwidth growth) than the analog technology proposed by other authors.

In this research, the environment was considered, namely: the devices used in the research process and the connections between them.

The results of measuring the bandwidth and the number of data transmission repetitions in the network were obtained.

An assessment of the reliability of the data was carried out – average values, maximum values, minimum values, mean square deviations, and errors of representativeness of the arithmetic mean.

The analysis of the obtained results was carried out – the balancing mechanism using the probability of getting into the connection channel is recommended for

solving the problem of network traffic balancing in small networks.

From further research perspectives, it is possible to consider the use of QoS measures together with balancing mechanisms of multiple communication channels to guarantee bandwidth for sensitive services and applications.

REFERENCES

[1] Liu X., Qian C., Hatcher W. G., Xu H., Liao W., Yu W. (2019). "Secure Internet of Things (IoT)-Based Smart-World Critical Infrastructures: Survey, Case Study and Research Opportunitie", in *IEEE Access, Vol. 8*, pp. 11825–11832. DOI: 10.1109/ACCESS.2019.2920763.

[2] Damasceno J., Dantas J., Araujo J. (2022). "Network Edge Router Performance Evaluation: An OpenWrt-Based Approach", *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, Madrid, Spain, pp. 170–173. DOI: 10.23919/CISTI54924.2022.9820027.

[3] Kafetzis D., Vassilaras S., Vardoulias G., Koutsopoulos I. (2022). "Software-Defined Networking Meets Software-Defined Radio in Mobile ad hoc Networks: State of the Art and Future Directions", in *IEEE Access, Vol. 10*, pp. 2305–2312. DOI: 10.1109/ACCESS.2022.3144072.

[4] Zhang P., Xie K., Kou C., Huang X., Wang A., Sun Q. (2019). "A Practical Traffic Control Scheme With Load Balancing Based on PCE Architecture", in *IEEE Access, Vol. 7*, pp. 1935–1942. DOI: 10.1109/ACCESS. 2019.2902610.

[5] Lemeshko O., Yevdokymenko M., Shapoval M. (2021). "Routing Model with Load Balancing on the Traffic Engineering Principles based on Information Security Risks", *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T)*, Kharkiv, Ukraine, pp. 114–126. DOI: 10.1109/PICST54195.2021.9772193.

[6] Torres R., Fortes S., Baena E., Barco R. (2021). "Social-Aware Load Balancing System for Crowds in Cellular Networks", in *IEEE Access, Vol. 9*, pp. 183–194. DOI: 10.1109/ACCESS.2021.3100459.

[7] Lim J., Yoo J., Won-Ki H. J. (2021). "Reinforcement Learning based Load Balancing for Data Center Networks", *IEEE 7th International Conference on Network Softwarization (NetSoft)*, Tokyo, Japan, pp. 14–29. DOI: 10.1109/NetSoft51509.2021.94925662.

[8] Binh L. H., Duong T. (2021). "Load balancing routing under constraints of quality of transmission in mesh wireless network based on software defined networking", in *Journal of Communications and Networks, Vol. 23, issue 1*, pp. 483–494, DOI: 10.23919/JCN.2021.000004.

[9] Karnani S., Shakya H. K. (2021). "Leveraging SDN for Load Balancing on Campus Network (CN)", *13th International Conference on Computational Intelligence and Communication Networks (CICN)*, Lima, Peru, pp. 324–338. DOI: 10.1109/CICN51697.2021.9574640.

[10] Pang S., Chen X., Zeng D. (2021). "Research on Dynamic Load Balancing of Data Center Network Based on SDN Architecture", *GLOBECOM 2020 – 2020 IEEE Global Communications Conference*, Taipei, Taiwan, pp. 324–338. DOI: 10.1109/GLOBECOM42002. 2020.9348059.

[11] Tsiunyk B., Muliarevych O. (2022). "Autonomous Face Detection System from Real-time Video Streaming for Ensuring the Intelligence Security System", *Advances in Cyber-Physical Systems, vol. 7, no. 2*, pp. 156–162. DOI: https://doi.org/10.23939/acps2022.02.156.

Klymyshyn Nazarii received his B.S. degree in Computer Engineering at Lviv Polytechnic National University, Ukraine, in 2022. He has professional experience working in IT since 2021 and currently working as a C/C++ software engineer in GloballLogic (Lviv, Ukraine).