Journal of Network and Systems Management (2022) 30:30
https://doi.org/10.1007/s10922-022-09642-y

Hybrid SDN Networks: A Multi-parameter Server Load Balancing Scheme

Teodor Malbašić[1] · Petar D. Bojović[2] · Živko Bojović[1] · Jelena Šuh[3] · Dušan Vujošević[2]

Received: 31 March 2021 / Revised: 11 September 2021 / Accepted: 2 January 2022 /
Published online: 27 January 2022

Abstract

Software-defined networking (SDN) provides many benefits, including traffic programmability, agility, and network automation. However, budget constraints burdened with technical (e.g., scalability, fault tolerance, security issues) and, sometimes, business challenges (user acceptance and confidence of network operators) make providers indecisive for full SDN deployment. Therefore, incremental deployment of SDN functionality through the placement of a limited set of SDN devices among traditional devices represents a rational and efficient environment that can offer customers modern and more data-intensive services. A unique challenge is the flexible distribution of loads on the servers that serve these services in network environments. The research in this paper focuses on developing a new load balancing scheme utilizing a hybrid SDN environment built with a minimal set of SDN devices (controller and one switch). We propose a novel load balancing scheme to monitor current server load indicators and apply multi-parameter metrics for scheduling connections to balance the load on the servers as efficiently as possible. The base of the new load balancing scheme is continuous monitoring of server load indicators and implementations of multi-parameter metrics (CPU load, I/O Read, I/O Write, Link Upload, Link Download) for scheduling connections. The testing performed on servers aims to balance the server's load as efficiently as possible. The obtained results have shown that this mechanism achieves better results than existing load balancing schemes in traditional and SDN networks. Moreover, a proposed load balancing scheme can be used with various services and applied in any client-server environment.

✉ Petar D. Bojović
petar.bojovic@paxy.in.rs

[1] Faculty of Technical Sciences, University of Novi Sad, 6 Trg Dositeja Obradovića, Novi Sad, Serbia

[2] The School of Computing, Union University Belgrade, 6/6 Knez Mihailova, Belgrade, Serbia

[3] Telekom Srbija, Bulevar umetnosti 16a, Belgrade, Serbia

Keywords SDN · Hybrid SDN · Load balancing · SNMP · Multi-parameter metrics

1 Introduction

Implementation of new intelligent services requires modernization of network infrastructure and more efficient use of available network resources. Software-Defined Networking (SDN), as a new network paradigm, decouples the network control (management) plane from the data plane, centralizing the whole control logic in an SDN controller [1]. It supports networks and smart services' dynamic nature while providing simple network management, enabling better Quality of Experience (QoE) for users and reducing costs [2]. Although SDN implies simplified hardware, software, and management, its full implementation requires replacing a large part of the existing traditional infrastructure. It is thus limited by budget [3] and with possible extensive downtime.

For these reasons, SDN is often deployed incrementally in large networks, building a hybrid infrastructure with a mix of a limited number of SDN-enabled and legacy devices. Two principal ways of hybrid SDN deployment exist [4]: a) Introduction of SDN switches in a legacy network and b) Implementation of hybrid SDN switches with both SDN and legacy switching functionalities. Besides cost reductions, hybrid SDN networks also provide certain SDN functionalities without implementing a full SDN network while enabling fine-grained traffic control. In that way, an SDN controller is free from tasks that traditional protocols can perform in hardware in a much more efficient manner. Sometimes it is necessary to interconnect two or more SDN networks via legacy network devices, and in that case, hybrid SDN must be used [5]. Our research aims to maximize the usage of existing elements of the traditional network and their functionality (e.g., SNMP for remote insight into the state of server resources) and apply a higher level of programmability (e.g., in the network traffic load balancing). The emergence of new traffic demanding services often leads to network resources overload and network congestion, resulting in reduced service availability and significantly affecting QoE. It seems necessary to overcome these challenges by implementing an adequate load balancing scheme. The load balancing in traditional networks is very demanding (both in cost and configuration) and primarily based on dedicated and highly expensive hardware (often a proprietary solution) [6]. The SDN concept can solve these challenges (reduce cost and increase load balancing efficiency) by implementing traditional load balancing schemes [7] or their modifications (e.g., based on the server response time in the server cluster [8]) in the software of SDN controller.

In their research, scientists use two approaches to solve the load balancing issue (the deterministic and the non-deterministic). Both approaches have certain disadvantages, which we will discuss later. Our solution aims to mitigate these disadvantages. We propose a new solution for dynamic load balancing in a hybrid SDN network—LBORU (Load Balancing by Optimizing Resource Utilization). The traditional algorithms (e.g., Random (R), Round-Robin (RR), or Weighted Fair Queueing (WFQ)) are relatively simple and do not use metrics based on the server's current resource occupancy. Our research focuses on a hybrid SDN network (with a

minimum number of SDN-enabled components) and the deployment of a new programmable load balancing scheme. This scheme uses a Simple Network Management Protocol (SNMP) mechanism for collecting server currently utilized resource information for the requests scheduling. We have performed this research in the Laboratory of School of Computing in Belgrade, Serbia. We have used an emulated virtualized network infrastructure as a testbed environment, added a POX SDN controller [9], and a hybrid SDN switch responsible for the load balancing in communication to the server cluster (other traffic is managed traditionally).

Our load balancing scheme starts from the observation that different services have different resource requirements and, for this reason, uses a metric based on multiple parameters (CPU load, I/O Read, I/O Write, Link Upload, Link Download). These parameters describe well the load on server resources, such as CPU, storage, and network links. We have evaluated our solution, comparing it to traditional methods and server response time-based load balancing scheme (LBBSRT) [8]. By ensuring a better balance in resource load, we have achieved improved system response. Our mechanism monitors the current CPU values and network resources load and makes better load balancing decisions than the existing mechanisms used for the comparison.

In our previous research [3], we presented a flexible network architecture for hybrid SDN infrastructure with a minimum number of SDN components (one SDN controller and one SDN switch per network [10]. Within such a very elastic environment, we have conducted research and accomplished the following contributions that we present in this paper:

- Design and implementation of a novel load balancing scheme based on the multi-parameter metric
- Proving the efficiency of our solution by comparison with traditional and LBBSRT load balancing methods.

The rest of the paper is organized as follows: Sect. 2 reviews the SDN concept, describes hybrid SDN network architecture and traditional load balancing schemes. Section 3 gives an overview of load balancing schemes in SDN networks. Section 4 details the design of our proposed load balancing scheme in a hybrid SDN network. Section 5 describes the implementation and presents a performance evaluation method. Section 6 gives test results and performance analysis. Section 7 presents results discussion. Finally, we conclude the paper with an overview of possible future work.

2 Background

Traditional network management's complexity and the large number of functionalities defined in the hardware lead to a lack of flexibility for solving users' requests and costly implementation. Migration to the SDN concept reduces computer network management's complexity and in some cases reduces network latency, as well as introduces a higher degree of reliability and programmability [11]. SDN achieves

this improvement using dynamic resource allocation following service requirements and fast and easy service implementation, the so-called "service provisioning on the fly" [12].

2.1 The SDN Concept

The SDN architecture has a three-layer structure, including application, control, and data [13] (Fig. 1). There are different user requirements, and it is necessary to implement various so-called northbound APIs on the application layer, providing different functionalities [14]. An SDN controller is a central part of the SDN architecture responsible for controlling and managing underlying infrastructure by offering network-wide visibility and direct control [15]. OpenFlow is the first standard protocol (southbound API) that uses the concept of flows for traffic identification based on matching rules defined by the SDN controller [16]. The SDN controller will create, either statically or dynamically, a rule for packet processing that updates SDN switches' flow tables and perform appropriate action accordingly. In that way, the SDN controller manages all the switches' flow tables simultaneously, enhancing the network performance and significantly reducing the network complexity [17].

Recently, SDN technologies have been deployed in different network environments (e.g., data centers, enterprise networks, 5G networks, Industry 4.0) to support the efficient implementation of network virtualization and Network functions virtualization (NFV). They allow administrators to control and manage their virtualized resources and network without detailed knowledge of the hardware technologies (simplified data plane and abstraction rather than specialized hardware) [18]. The process of virtualization introduces a higher degree of scalability and security in data centers. It enables the automation of virtual machine migrations, positively

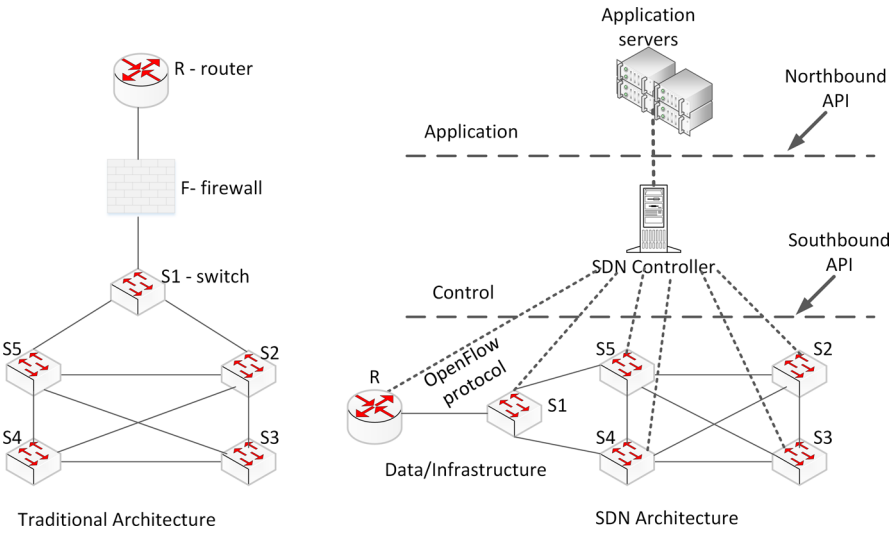

Fig. 1 Traditional vs SDN architecture

affecting the utilization of server and network resources [19]. The SDN applies well-defined data plane abstraction, including packet forwarding abstraction models, circuit switching abstraction models, wireless integration, and an evolved packet core support [18].

With NFV, functions that traditionally run on dedicated equipment (e.g., firewalls and load balancers) can be virtualized and placed closer to the desired point in a network, containerized with an application. NFV allows multiple logical networks to run on common physical infrastructure. In [20], the authors propose a new resource allocation for industry 4.0 based on SDN and NFV technologies, machine learning tools, and network slicing depending on service requirements regarding bandwidth, delay, and reliability. In [21], besides network slicing, the authors point to an increase in the Industrial Internet of Things (IIoT) computational capacity and propose architectural improvements to efficiently accommodate diverse QoS demands on shared network infrastructure, such as data privacy. They recognize that federated reinforcement learning (RL) has become a promising approach that distributes data acquisition and computation tasks over distributed networks exploiting local computation capacities and agents' self-learning experiences. They propose a novel deep RL scheme to provide federated and dynamic network management and resource allocation for differentiated QoS services in IIoT networks.

2.2 The Hybrid SDN Concept

The technical, financial, and business challenges accompany SDN's full implementation. The hybrid SDN infrastructure can be an intermediate solution, representing the integration of legacy and SDN devices working in parallel and taking advantage of traditional and SDN networks. It is necessary to develop an incremental deployment strategy to implement such an infrastructure [22]. Hybrid SDN networks consist of both centralized and decentralized architectures that must communicate with each other to obtain optimal performance and QoE through network configuration, control, and management. Hybrid SDN architecture relies on a 3C model [22]: a) Coexistence—heterogeneity of infrastructure in the data plane, control plane, or both data and control plane; b) Communication—the interaction of legacy and SDN devices in order to enable understanding and functionality sharing and distribution at data plane, control plane or both data and control plane; c) Crossbreeding—a combination of different network paradigms with complementary characteristics to minimize budget, simplify the transition, automation, and traffic policy definition, and provide high-level scalability and robustness.

In [23], four hybrid SDN models are defined (Fig. 2):

- Topology-based hybrid SDN model—This implies partitioning the network into zones where each device can be a member of only one zone (legacy or SDN). A zone is composed of multiple devices controlled by the same network paradigm.
- Service-based hybrid SDN model—Legacy and SDN devices provide different services. For end-to-end services, it can be necessary that two network paradigms control a set of devices concurrently (controlling a different portion of the

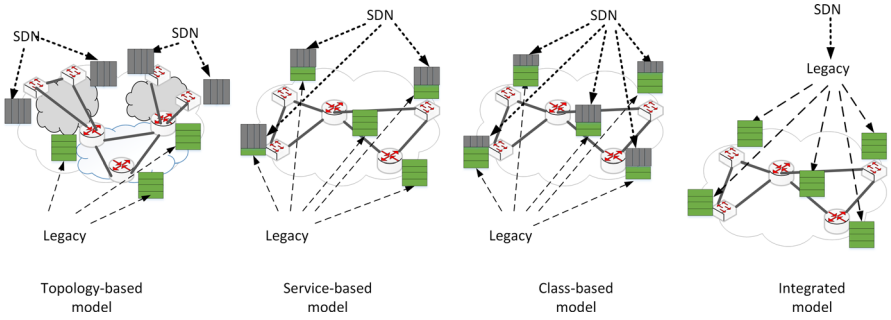

Fig. 2 Hybrid SDN models, based on the classification in [22]

Forwarding Information Base—FIB) of each device) and, at the same time, some devices can be controlled by only one network paradigm.

- Class-based hybrid SDN model—This implies partitioning the network traffic into classes that can be controlled by legacy or SDN paradigm. All network devices usually have both legacy and SDN functionalities.
- Integrated hybrid SDN model—SDN is responsible for all the network services, and it uses legacy network protocols as an interface to device FIBs. A legacy paradigm that is controlled by the SDN controller manages the device FIBs.

The implementation of a hybrid SDN infrastructure has certain limitations [22]. One of the limitations is the heterogeneous control plane's management complexity, where the reconfiguration process causes forwarding inconsistency and, consequently, can provoke forwarding loops and traffic black holes [24]. The other limitation is data plane complexity, where a translation of protocol from legacy to SDN and vice versa leads to performance degradation related to latency and processing time. SDN controller limitations are related to implementing two or more controllers within the hybrid SDN infrastructure (which can cause a problem with latency, traffic engineering, scalability, and security). Implementing adequate traffic engineering methods can also be a big challenge, mainly if each device does not support flow abstraction [25]. Nevertheless, hybrid architectures can be a suitable solution for achieving the required QoS (Quality of Service) levels in convergent networks [26].

The deployment of hybrid SDN infrastructure includes the placement of SDN devices in a traditional network. In [27], the Panopticon approach creates a hybrid SDN network by interconnecting legacy and SDN devices with the idea to force all traffic to traverse an SDN switch controlled by the SDN controller. Although not all networking aspects (e.g., traffic engineering and load balancing) were considered, this research showed a performance increase compared to the original network. Internet service providers can also use the advantages of incremental deployment of SDN network infrastructure to achieve a higher level of programmability and traffic engineering [28]. They classify network traffic into programmable and non-programmable depending on whether the traffic traverses through an SDN switch. Sometimes, deployment of hybrid SDN infrastructure implies adding SDN devices instead of replacing legacy devices [29]. The SDN devices connect to legacy

devices, and both distributed routing protocols and the SDN controller process the traffic. The SDN functionality can also be implemented by installing SDN shim hardware in legacy devices [30] to enable communication between an SDN controller and a modified legacy device. In [31], the authors present a service-based hybrid SDN wireless network model, which gives better performance, especially related to resiliency to path failures.

In practice, there are many other interesting scenarios of the hybrid SDN implementation. A hybrid SDN based architecture for vehicular ad hoc networks provides end-to-end data transmission with flexibility, scalability, and programmability features. It can solve problems in traditional traffic management systems with variable traffic network conditions [32]. For a compute-intensive scenario in a framework dedicated to computing and analysis of data on the wire", i.e., while the data is still in transit, researchers plan to achieve acceleration using a hybrid SDN testbed with graphics processing units and field-programmable gate arrays [33].

2.3 Traditional Load Balancing

Network overload problems can be mitigated by applying different load balancing schemes and improving network efficiency and reliability (e.g., increasing scalability and throughput, decreasing response time and resource consumption, avoiding overload of any single resource) [34]. Static or dynamic traffic balancing, or a combination of both, can be implemented using some of the four traditional load balancing schemes [35]. The first scheme refers to the clients that collect server running parameters to schedule requests to different servers and achieve a certain load balancing level. The second scheme is based on the reverse proxy server deployment on the middle layer, combining the load balancing technology with the caching technology to enhance the access speed. The third scheme is DNS-based, where multiple IP addresses in the server clusters can use a single domain (domain name servers use a pooling method to schedule clients' requests to different servers). Despite this scheme's simplicity, the DNS server is not aware of the difference among the servers, and it cannot reflect the current state of the servers [36]. The load balancing at the transport layer represents the fourth scheme based on the communication between the clients and the load balancing server, which forwards clients' requests to the backend servers according to policies (e.g., Linux Virtual Server [37]). This approach is efficient but expensive since it often requires additional hardware.

3 Related Work

3.1 SDN Load Balancing

The SDN controller, which makes the forwarding decision for every new flow, has a crucial role in SDN load balancing and is responsible for Real-time Least loaded Server selections (RLSs) [38]. However, centralizing logic in a single controller can be a problem regarding responsiveness, reliability, and scalability [39]. A

deterministic approach to load balancing in the SDN always gives the same output for a specific input, whereas the values of parameters and the initial situations make the output. A non-deterministic approach represents an algorithm that has different results on different runs for the same input. The authors in [34] summarize both approaches' main advantages and disadvantages (Fig. 3).

Regardless of the load balancing approach implemented in the network, the appropriate metrics must be defined. Various qualitative parameters can be used for load balancing scheme evaluation in SDN [34], e.g., resource utilization (choice of optimal load balancing algorithm can maximize resource utilization [40]), latency (e.g., direct routing based load balancing algorithms minimize latency [41]), packet loss rate, response time, throughput, migration cost, workload (load balancing of workloads among controllers [42]), energy consumption (choice of suitable load balancing scheme can reduce energy consumption [43]), forwarding entries (it is necessary to decrease the number of forwarding entries to save memory resources [44]), and execution time. Interesting research about the SDN bringing a higher level of dynamics and programmability to the network describes load balancing using the algorithms for client traffic partitioning based on load balancing weights [45]. Given the growing complexity of infrastructure and traffic volume, in [46], the possibility of multiple load balancer deployment for different services is discussed (e.g., one for web traffic and the other for email). In [47], the authors point to a load imbalance state in the network that may occur when there are multiple SDN controllers and can notably degrade service levels in some parts of the network. They propose a new load balancing scheme based on the integer linear programming technique (ILP). The authors in [48] propose a load balancing scheme in SDN and implement a new algorithm introducing the load-driven penalty concept to optimize the switch-to-controller assignment problem and achieve a trade-off between the round trip time and the controller load.

3.2 Load Balancing in Hybrid SDN Networks

Given that the deployment of SDN is not possible in the short term and that hybrid SDN represents a reasonable solution, we opt that solving the load balancing issue

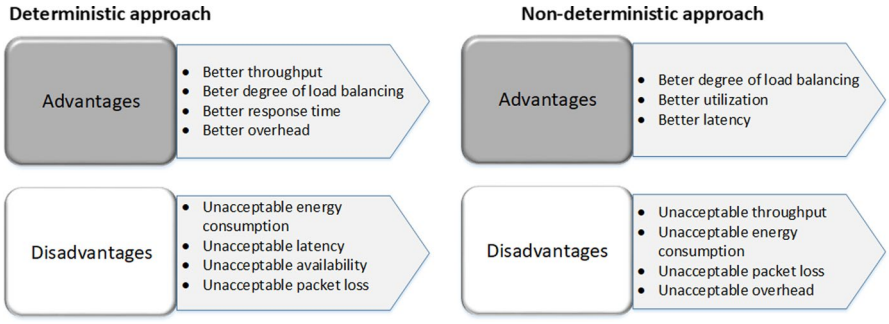

Fig. 3 Main advantages and disadvantages of the deterministic and non-deterministic approaches to the load balancing in the SDN, based on the classification in [31]

should focus on hybrid SDN networks. In [49], the authors propose an innovative routing and flow scheduling method using link utilization and CPU time as metrics. This approach has some limitations since it is focused only on solving the problem of Open Shortest Path First (OSPF) network congestion. The improvement of traffic flow management is the subject of research in [50]. The authors suggest flow management based on a splitting mechanism where the path for every flow must go through the SDN switch to achieve better flow control and traffic engineering (they use link utilization and latency as metrics). Since these approaches do not consider the QoS, they can result in routing process inefficiency.

As can be seen from the previous approaches, no load balancing scheme uses all QoS parameters. Therefore, it could be interesting to investigate how to choose QoS parameters to optimize traffic balancing decisions in different service scenarios. Some authors go further and investigate the possibilities of hybrid SDN deployment within data centers. To realize efficient traffic forwarding, the authors in [51] use QoS aware routing with metrics based on packet loss, delay, and bandwidth. The main constraints of this approach are high complexity regarding computation and possible processing delay increase.

A significant limitation in many load balancing schemes is that they do not include continuous detection of resource load in the decision-making process, which is essential considering the nature of new communications (e.g., IoT and M2M) and the growing trend of Internet users. The reason is that these proposed methods balance traffic using traffic metrics (e.g., link metric, bandwidth, and RTT) and do not consider service resource requirements. Our goal is to develop a load balancing scheme that we can customize regarding service resource usage.

4 Novel Load Balancing Scheme

The traditional networks commonly use simple traffic distribution mechanisms such as RR or WFQ algorithms. They do not consider the server's resource utilization but only allocate traffic based on the previous connection distribution. Thus, they can not achieve optimal network performances in terms of scalability and reliability. By implementing programmable logic for network traffic management, the possibility of retrieving external information such as, e.g., server load, thus increasing the efficiency of the decision in balancing traffic. Our research introduces a minimum set of components (e.g., SDN-aware switch and SDN controller) in the traditional network to keep an acceptable implementation cost. In this way, we can apply new technologies and enable a higher level of programmability to realize load balancing. Although the solution we propose contains a minimum set of devices, we can apply it in any production environment. There it is necessary to implement redundant components to provide the system's required fault tolerance.

4.1 A Proposed Hybrid SDN Architecture

As shown in Fig. 4, we propose replacing the switch connecting to servers with the SDN-aware switch for partial implementation of the SDN functionality. Besides using existing traffic forwarding methods, we have created conditions for traffic forwarding based on the SDN controller instructions (using OpenFlow protocol). The implementation of SDN functionality aims to implement a scheduling mechanism that, in combination with existing technologies, collects real-time information about server load and, using this information, performs scheduling tasks and balances traffic. Thus, the SDN switch performs tasks related to load balancing in the SDN part of the network, while all other traffic is processed traditionally. We can apply the proposed architecture in any client-server environment.

It is essential to highlight that the SDN controller examines the servers' loads via the SNMP protocol to identify the lowest load one. The controller's task is to form flow instructions after processing each first packet. Following the instructions, the SDN switch should forward the traffic. In other words, the SDN controller decides on which server the SDN switch should route all future packets of the same connection.

The solution shown in Fig. 4 aims to enable a more optimal distribution of user requests by applying dynamic network control and make the network itself more agile. Based on multi-parameter load analysis on servers, this approach should support a more significant number of simultaneous user requests and reduce response time on different servers in LAN networks and Data Centers.

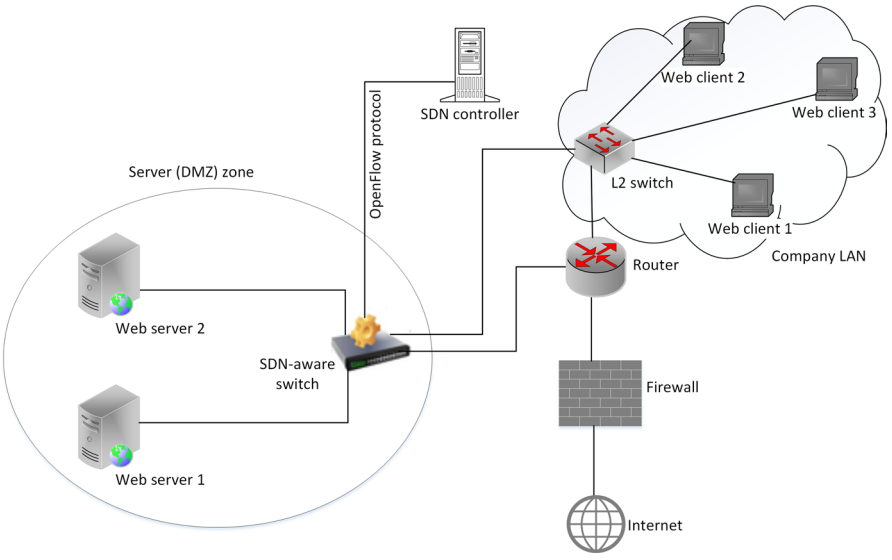

Fig. 4 The example of hybrid network infrastructure with a minimum number of SDN components

4.2 Multi-parameter Load Balancing Scheme

The load balancing scheme that we propose requires a new protocol to regulate communication between clients and servers in a hybrid SDN network. The main idea is that the SDN controller allocates a single virtual IP (*vIP*) and MAC address (*vMAC*) for all servers whose traffic needs to be balanced. To avoid the potential problem of ARP caching, which can impair load balancing efficiency, we have provided persistent mapping of a *vIP* address to a *vMAC* address. This *vIP* address, as a destination address in clients' packets, is the address to which clients send requests for specific resources. When it comes to new connections, the SDN switch must forward requests to the SDN controller responsible for requests forwarding to a specific server (for existing connections, the SDN switch forwards packets according to existing records in the flow table). In this way, the SDN controller directly implements the load balancing functionality (Fig. 5).

The SDN controller decides to forward a specific connection by accepting the first packet of that connection (OpenFlow PacketIn), identifying the server with the least load, and then creating the appropriate flow instructions (OpenFlow PacketOut). Following these instructions, the SDN switch must:

1. modify the packet and change the destination *vIP* and *vMAC* address to the real addresses (*rIP* and *rMAC*) of the server with the least load, and forward the modified packet to the appropriate server;

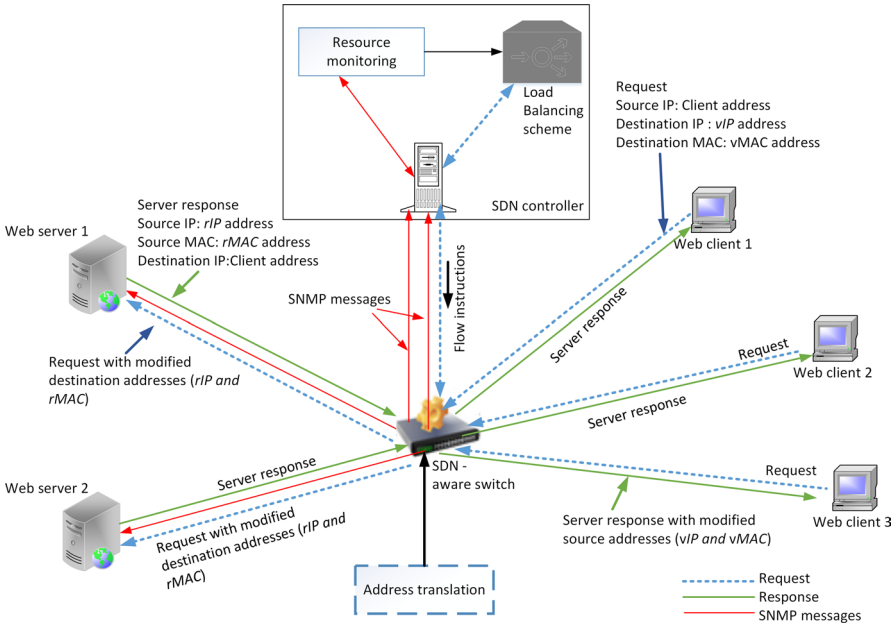

Fig. 5 Novel load balancing scheme

2. upon receipt of the reply packet from the server, the server's real address (source packet parameters) is retranslated into *vIP* and *vMAC*, and the packet forwarded to the client.

The SDN switch caches the translation information for a certain period. Each subsequent packet of the same connection is processed directly on the SDN switch following the flow table records. The SDN controller has two processes: collecting information about server load via the SNMP protocol and making decisions about traffic forwarding to a specific server. The messages exchange flow diagram is shown in Fig. 6.

The SDN controller collects information about server load using the following algorithm: The round-robin mechanism selects one of the defined servers. The SDN controller checks servers' SNMP availability, and if it is not available, it moves on to the next one. The SDN controller collects CPU load, I/O Read, I/O Write, Link Upload, and Link Download data via SNMP protocol and stores them in a local matrix variable. SNMP polling intervals should enable us to acquire resource load data often enough to ensure proper load balance. However, polling intervals could introduce unnecessary overhead load on the server too often. We propose a one-second interval as a balance between speed of resource usage evaluation and SNMP polling overhead. In order to obtain faster resource balancing, we could use lower

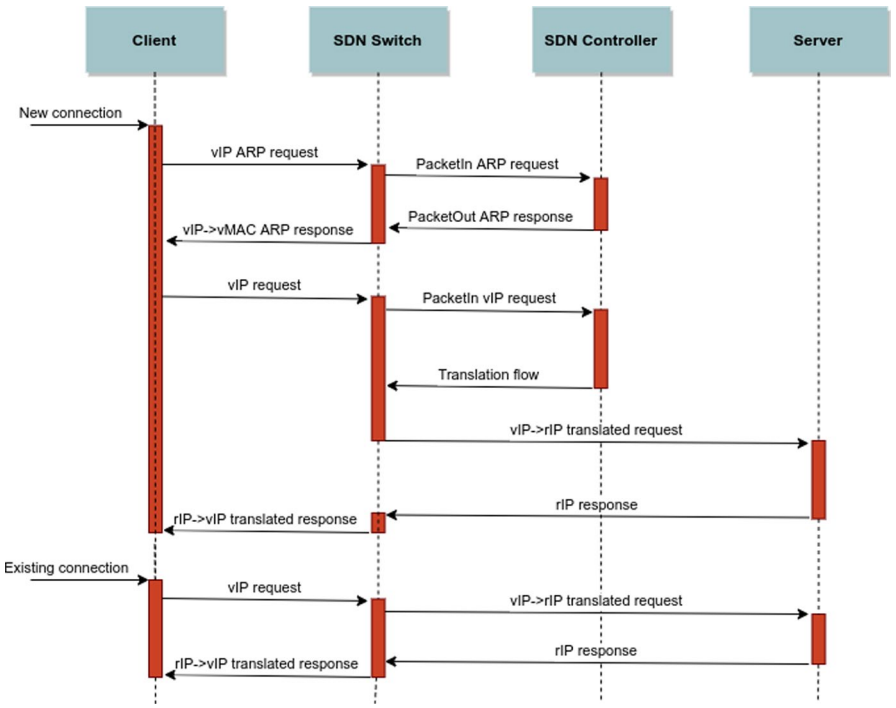

Fig. 6 Connection flow diagram

polling intervals. In this way, the SDN controller can perform a fast performance comparison with other available servers (Algorithm 1).

Algorithm 1: SNMP Algorithm

```
Input  : server - set of servers (S_1, S_2, ..., S_N)
Output: metrics - set of metrics for each avalable
          server
for server = S_1, S_2, ..., S_N do
    if SNMP_is_avalable (server) then
        cpu=SNMP_get (server, CPU_OID);
        ioread=SNMP_get (server, IOREAD_OID);
        iowrite=SNMP_get (server, IOWRITE_OID);
        download=SNMP_get (server,
          DOWNLOAD_OID);
        upload=SNMP_get (server, UPLOAD_OID);

        metrics(server)=(cpu,ioread,iowrite,download,upload);
    end
end
```

To decide which server to forward the connection to, the SDN controller needs to compare the load values of the corresponding parameters for each server. For this comparison, it is necessary to form a matrix A (1) from the collected parameters whose elements are a_{ij}, where i represents the ordinal number of the parameter (parameter 1—CPU, parameter 2 - I/O Read, parameter 3—I/O Write, Parameter 4—Link Upload, Parameter 5—Link Download), and j is server number ($j = 1, ..., n$):

$$A = \left\| a_{ij} \right\|_{(5*n))} = \begin{pmatrix} a_{cpu_1} & a_{cpu_2} & \dots & a_{cpu_n} \\ a_{ior_1} & a_{ior_2} & \dots & a_{ior_n} \\ a_{iow_1} & a_{iow_2} & \dots & a_{iow_n} \\ a_{up_1} & a_{up_2} & \dots & a_{up_n} \\ a_{dw_1} & a_{dw_2} & \dots & a_{dw_n} \end{pmatrix} \tag{1}$$

For each parameter, it is necessary to find a server or servers with a same, minimum value and assign a value according to (2):

$$b_{ij} = \begin{cases} 1, \ a_{ij} = \min\limits_{j=1,\dots,n}(a_{ij}) \\ 0, \ a_{ij} > \min\limits_{j=1,\dots,n}(a_{ij}) \end{cases} \tag{2}$$

By applying this relation, we obtain a matrix B (3) with elements bij representing the points that are assigned to the servers for each parameter:

$$B = \left\| b_{ij} \right\|_{(5*n)} \tag{3}$$

Each row in matrix B can have one or more bij elements, whose value is different from zero (multiple servers have the same, minimum value of load aij for the

observed parameter). The goal is to find the server with the least load at a given time. Thus, we must emphasize that each parameter (CPU, I/O Read, I/O Write, Link Upload, Link Download) does not have an equal impact on every service. Therefore, it is necessary to valorize the impact of each parameter. In other words, depending on the type of service, we can define weight factor k_i for each parameter. So, element bij in matrix B is multiplied by k_i, such that $k\ i$ [0,1] for every $i=1,...,5$. The default k_i value is 1 and could be modified to adjust the parameter impact. For example, in the case of database load balancing, we would define greater significance on k factor for I/O read and write usage than on Download/Upload parameters (k values vector as following—CPU=0.5, I/O Read=1, I/O Write=1, Upload=0.2, Download=0.2) to obtain better database performance and responses.

The load balancing scheme defines that the server with the least load is the server with the most points. Therefore, it is necessary to calculate the total number of points for each server by relation (4):

$$S_j = \sum_{i=1}^{5} k_i b_{ij}, j = 1, ..., n \tag{4}$$

By applying relation (4), the vector is obtained: $Sj = [S1, S2, ..., Sn]$. It is necessary to perform maximization by finding the element W (index of the server), with the maximal value using the relation (5):

$$W = \left\{ j \mid S_j = \max_{m=1,...,n}(S_m) \right\} \tag{5}$$

Thus, by obtaining information about the server's index with the least current load, the SDN controller becomes aware of the *rIP* address it should forward the traffic to and generates instructions for the SDN switch to perform translation from *vIP* to *rIP* address.

5 Implementation and Evaluation

We have defined a testing scenario and created a testbed environment that realistically reflects the traditionally organized network with an on-site data center to effectively implement and evaluate the proposed load balancing scheme in a hybrid SDN network. Our research has used a virtual environment emulation method, with virtualized hardware, while the software applied has been the same as it would be on physical devices.

5.1 Testbed Environment

We have used the EVE-NG platform [52] to build a testbed environment with virtual servers and other network resources and create a proof of concept. We have implemented the following virtual components (Fig. 7):

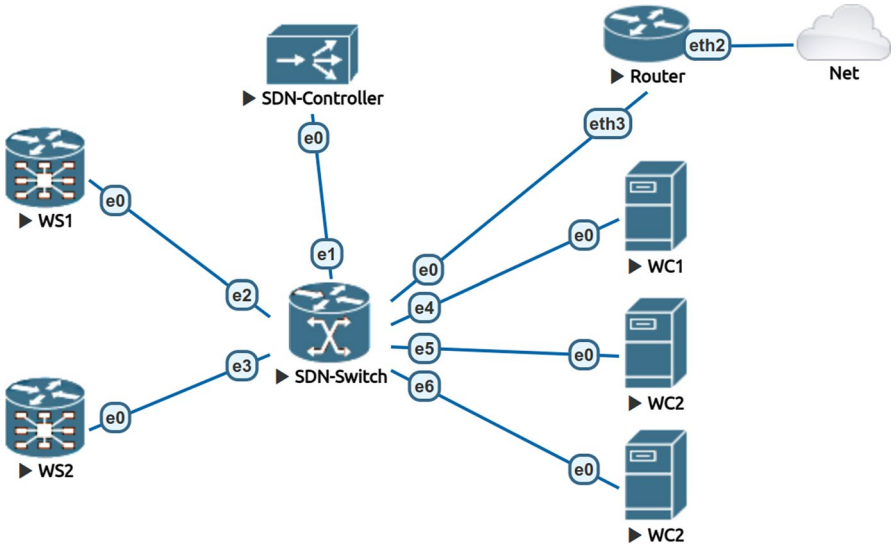

Fig. 7 EVE-NG testbed environment

- SDN-Switch—an Linux OVS router with the role of SDN-aware switch
- Six virtual machine instances (VM) with Linux Ubuntu servers
- Web Server 1 (WS1), Web Server 2 (WS2)—VM instances for Web Servers
- Web client 1 (WC1), Web client 2 (WC2), Web client 3 (WC3)—VM instances for web clients
- SDN-Controller—VM instance with POX SDN controller
- Router—an instance of the Mikrotik ROS router with L3 functionality
- Net—Internet connection.

Today's implementation of most companies' services uses web communication, which offers high flexibility by relying on the web as a mature technology. Web technology enables dynamic scaling of resources and supports a high level of service performance. It is necessary to enable dynamic traffic balancing between servers having the same function to perform dynamic scaling of resources. Therefore, we have set up two web servers in our testbed environment and implemented a dynamic load balancer for scheduling connections of multiple web clients.

As shown in Fig. 7, we have connected all virtual machines (with Linux OS) to the SDN switch, with the Open vSwitch (OVS) platform [53] managing flows via the OpenFlow protocol. We have implemented a router with DHCP server functionality to enable dynamic and persistent IP addresses assignment. In the emulated network, we have had to install the appropriate software on virtual machines (e.g., Java, Apache, SNMP) and connect the router to the public Internet via the Net interface. The physical host with the emulated environment forwards packets through real physical networks.

We have configured the OVS in a way that we merged all SDN switch interfaces into one virtual SDN bridge ('br-sdn'). As a network entity, this bridge obtains its IP

address via a DHCP from the router. Before the IP address on the SDN controller is configured, OVS behaves like a traditional switch. Thus, the OVS performs the L2 forwarding process of populating flow tables based on the original MAC address and forwards the frames based on the learned flows, and if necessary, it also floods the frames. From the moment the IP address is configured on the SDN controller, the SDN switch is forwarding every packet to the SDN controller for further processing due to missing instruction in the flow table.

The virtual machine with SDN controller functionality has a control function in this virtual environment. Various OpenFlow controllers have been publicly released (e.g., POX, NOX, Floodlight, IRIS, OpenDaylight, Jaxon, NodeFlow, Helios) [54]. We have decided to implement a POX controller [9] in Python because it has simple implementation and support for many libraries. We have developed a POX controller module to implement the functionalities described in Sect. 4.

5.2 Hybrid SDN Implementation

From the moment the OVS connects to the SDN controller, the SDN controller processes all new OVS flows. The SDN controller must perform proactive flow insertion to edit the traffic forwarding policy and define which of the flows will be managed by the SDN controller and which traditionally. Specifically, these are the proactive flows:

- (P1) ARP → PacketIn, Normal packets sent to a broadcast address will be forwarded to the SDN controller and then processed in the traditional way
- (P2) DstIP: *vIP* → PacketIn, PRI: 1000 packets sent to *vIP* will be forwarded to the SDN controller unless there is a higher priority flow (smaller number)
- (P3) Any → Normal, PRI: 500 all other packets are traditionally processed unless there is a higher priority flow.

The Normal instruction aims to indicate to the SDN device that the flow should be processed traditionally. The flow processing depends on the SDN device's primary function, i.e., whether the SDN functionality executes on a switch (L2 forwarding) or a router (L3 routing process).

The fact is that proactive flows can define rules, which change the OVS SDN bridge's default behavior. Therefore, the idea is to send to the SDN controller only packets with destination *vIP* address (P2) and ARP packets (P1). All other packets will be automatically processed by traditional L2 forwarding on the SDN switch itself without the controller actions (P3) unless there is a higher priority flow.

5.3 ARP Handling

We need to resolve the *vIP* address to the *vMAC* address to implement the load balancing method proposed in Sect. 4. Before sending the packet to the *vIP*, the web server must have information about the destination MAC address. For this reason, it sends an ARP request for resolving the *vIP* to the *vMAC*. The SDN controller

must also receive this ARP request (RA). At this request, the SDN controller forms a PacketOut with an ARP response containing a predefined mapping of the *vIP* address to the *vMAC* address and forwards it to the SDN switch. Upon receiving a PacketOut with an ARP response, the SDN switch forwards the ARP response to the port from which the ARP request has come. After the server receives the ARP response for the *vIP*, it can populate the destination MAC address with the *vMAC* and prepare the packet to send.

Thus, the SDN controller forms a reactive flow in this way:

- (RA) PacketOut(ARP response, DST: ARP req SRC MAC) → DPort: Req SPort.

We should mention that the SDN controller ignores any other ARP request because these requests will be processed traditionally (flooding).

5.4 Load Balancing Procedure

As discussed in Sect. 4, the packet sent to the *vIP* arrives at the SDN switch, which, since it is a new connection, still does not have a reactive flow instruction in its flow table. Therefore, it executes the proactive instruction (P2), sends the packet to the SDN controller, performs the load balancing procedure, and decides which server *rIP* address it should use for this request. The SDN controller creates two reactive flows:

- (R1) DstIP: *vIP*, SrcIP: *srcIP*, SrcTCPPort: *srcTCPPort* → Modify(DstIP: *rIP(srvN)*, DstMAC: *rMAC(srvN)*), Normal, PRI: 1500
- (R2) SrcIP: *rIP(srvN)*, DstIP: *srcIP*, DstTCPPort: *srcTCPPort* → Modify(SrcIP: *vIP*, SrcMAC: *vMAC*), Normal, PRI:1500

with the following elements:

- *srcIP*—*rIP* client address (in our case *rIP* address of Web1, Web2, Web3—Fig. 7)
- *srcTCPPort*—client's application source port (a different port for each new TCP connection)
- *srvN*—the ordinal number of the server where the request is to be forwarded (e.g., 1 for WS1 and 2 for WS2)
- *rIP(srvN)*—*rIP* address of the server where the request is to be forwarded
- *rMAC(srvN)*—*rMAC* address of the server where the request is to be forwarded.

The reactive flow instruction (R1) represents the response of the SDN controller to the PacketIN request. It instructs the SDN switch to match the packet of a particular connection sent to the server's virtual IP address and then modify it by replacing the virtual destination address parameters (*vIP* and *vMAC*) with the real server parameters to which the request should be forwarded. After modification, the switch forwards the packet using the existing L2 forwarding mechanism. By defining a higher

priority (1500) for the reactive flow instruction (R1), compared to the proactive flow instruction (P2), each future packet with the same parameters (source IP address, port, and *vIP* destination address) will be forwarded immediately without SDN controller action.

Since the server's response should be returned to the request's sender, the SDN controller with the reactive flow (R2) instructs the SDN switch to modify each packet coming from the observed server by replacing *rIP* and *rMAC* server addresses with virtual parameters (*vIP* and *vMAC*). The reactive flow instruction (R2) priority is higher than the proactive flow instruction (P3) priority, so the switch will forward the modified packet by the L2 forwarding mechanism to the sender without SDN controller action.

5.5 Testing Methodology

The proposed load balancing scheme evaluation in hybrid SDN networks has aimed to gain an accurate insight into its capabilities. Therefore, it has been necessary to perform the testing in conditions that largely correspond to the real environment situation. Given the increase in web services, we have decided to check the new load balancing scheme's efficiency on web servers. We have tested the proposed mechanism in an emulated environment with virtual client instances, web servers, and SDN components (SDN controller and hybrid SDN switch). To obtain the most reliable results, we have performed testing in a controlled manner, gradually including an increasing number of users connections to maximize the load of the web servers and check the effectiveness of several load balancing mechanisms, including the proposed LBORU method.

For this reason, we have developed a testing tool that sends web requests and consists of two components:

- Job component—Establishes a connection with the web server, executes a single request transaction, and reports on the time required for its completion. The Job component procedure is executed in an infinite loop to simulate a single competing user, constantly performing transactions.
- Concurrency component—Increases the number of parallel users in a controlled manner, records the response time for each transaction, and performs statistical calculations.

The Job component has been sending web requests, while the Concurrency component has been collecting testing parameters such as:

- The number of concurrent connections
- Average time required to complete the single request
- The number of completed transactions per second.

To get a realistic insight into the proposed solution's performance, instead of commercial web servers (e.g., Apache or NginX), which already have some of the

mechanisms to optimize the use of their resources, we have created our web server. Our goal has been to obtain the interpretable measurements, which will be useful regardless of the optimization mechanisms' existence. This server is written in the Java programming language and does not have any resource optimization mechanisms (all tools developed for testing purposes are available on GitHub [55]. The goal of testing has been to load the created JAVA servers with a large number of service requests. We have started by observing that various services require a different level of server resource engagement (e.g., CPU, RAM, and network resources) during the response generation process. In other words, the response generation is loading Java servers depending on the requests.

We have compared other commonly used load balancing schemes such as Random, Round Robin, and LBBSRT to gain a clear insight into the proposed scheme's possibilities. In the case of Random or Round Robin load balancing, which are implemented directly on several web clients, the requests come through Random distribution from the server's perspective (web clients do not know how much each server is loaded). We have introduced the SDN controller and applied the Round Robin, LBBSRT, and LBORU mechanism, to create conditions for centralized control of client requests and connections scheduling in front of the web servers.

6 Test Results and Performance Analysis

We perform the testing in a realistic environment with virtual machines in the EVE-NG testbed environment (6 repetitions). There is an element of temporal load variations in the quality of the results, which indicates that increasing the number of repetitions would only introduce noise into the results. In the initial testing phase, we will analyze the influence of processing overhead on each first packet and identify the value of an initial processing delay in the SDN network. After that, we will analyze the relationship between the number of transactions and concurrent clients (connections). This analysis aims to identify how the SDN concept's initial delay affects the transaction number and average transaction time. Also, we must assess the efficiency of our load balancing scheme and analyze servers' resource load disbalance (primarily CPU load) as one of the efficiency indicators for load balancing mechanisms.

For an objective analysis of the obtained results, we must consider that direct connections from web clients to available web servers have a certain advantage over the SDN approach because of initial packet processing in SDN connections. The SDN switch must send this packet to the SDN controller to process it and decide (give instruction) on further forwarding [1]. Therefore, we assume that the direct connections scheduled by the Random load balancing mechanism are the baseline for the performance evaluation.

Figure 8 shows the influence of processing overhead on each TCP SYN packet in SDN (the first packet of each connection), through an initial processing delay of approximately 1000 ms. The graph shows the results for 30 transactions in the same connection. It is important to note that each subsequent packet's processing delay has a similar value as for the direct connection.

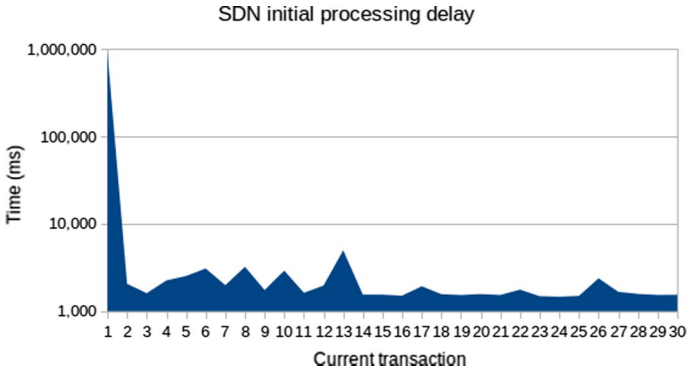

Fig. 8 Initial processing delay of SDN controlled connections

We analyze the results obtained by testing the previously described load balancing mechanisms (R, RR, LBBSRT, and LBORU) in the environment described in section 4. One of the key parameters analyzed is the number of transactions and their dependence on the number of concurrent clients (connections), as shown in Fig. 9. The conclusion is that despite the initial delay present in the SDN concept, the Random mechanism did not have a significant advantage in terms of transactions number compared to the SDN. There is a linear dependence between the number of transactions per second and the number of concurrent connections up to 40 concurrent connections. Further increase in the number of concurrent connections leads to losing this dependence due to resource overload on both servers. Once the resource overload occurs on servers, there is no significant advantage of any tested load balancing mechanisms.

Analysis of the average transaction time (Fig. 10) shows that the first packet's overhead in the SDN concept does not have a significant impact.

To gain insight into each of the tested load balancing mechanisms' efficiency, we have analyzed the data describing a server load. Thus, Fig. 11 shows the CPU

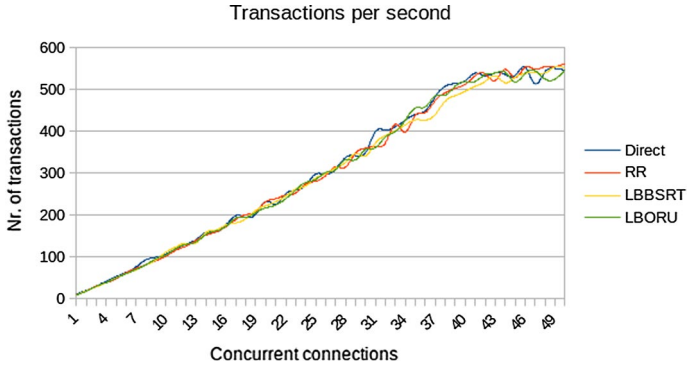

Fig. 9 Transactions per second comparison

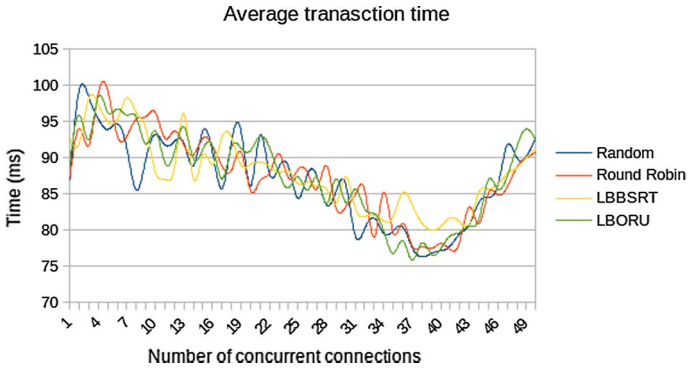

Fig. 10 Average transaction time comparison

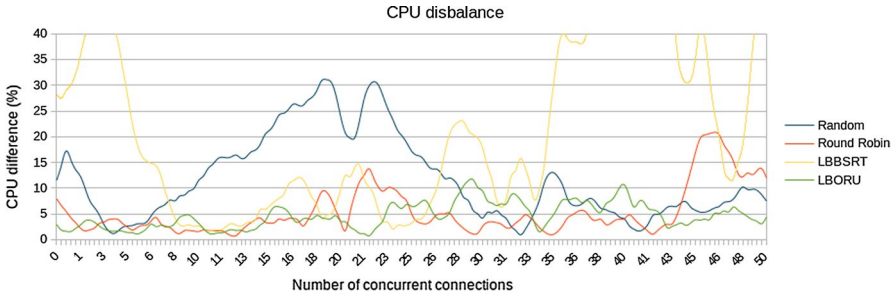

Fig. 11 Comparison of CPU disbalance

load disbalance (CPU load difference between WS1 and WS2) as one of the efficiency indicators for load balancing mechanisms. The smaller disbalance means a more uniform CPU load on servers.

The Random load balancing causes a significant CPU disbalance, primarily because the web client is not aware of the CPU resource load on the web servers. However, with the increase in the number of concurrent connections, it comes to server overload, and then the disbalance becomes less noticeable. On the other hand, the Round Robin mechanism has shown satisfying results for CPU load balancing up to the moment of server overload. Due to the inconsistency of individual requirements regarding the engagement of CPU resources generating responses on web servers, it is noticeable that this mechanism creates an increasing disbalance with the increase in the number of concurrent connections. To accurately analyze the results obtained using the LBBSRT mechanism, we must consider that this mechanism relies on the value of the RTT parameter. The fact is that measuring the RTT parameter's value via the ICMP protocol requires little resources on the server (there is a quick response to each server query). However, the test results indicate that the LBBSRT mechanism cannot detect disbalance in a sufficiently precise way and timely equalize the CPU load on web servers.

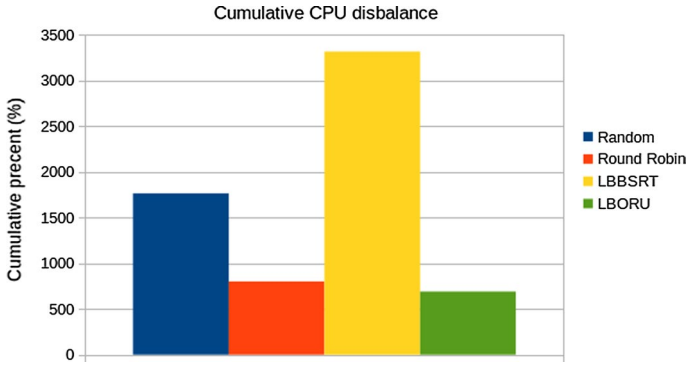

Fig. 12 Cumulative CPU disbalance

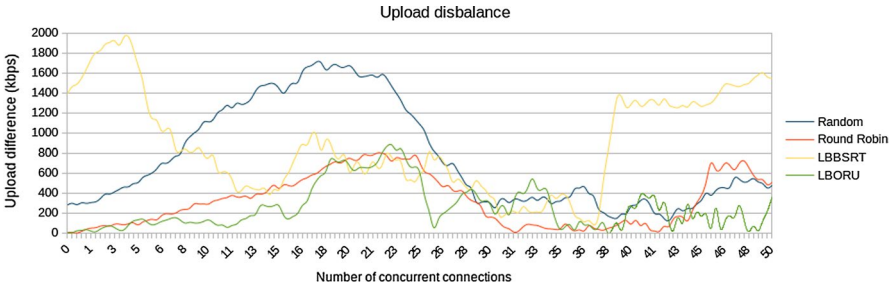

Fig. 13 Comparison of upload disbalance

The mechanism proposed in this paper (LBORU) monitors several operating system parameters. The decision to forward requests to a specific server is based on these parameters' values in real-time, collected using the SNMP protocol. The obtained results indicate a maximum CPU load disbalance of about 10%, which can be considered very low. More importantly, such value of CPU disbalance remains within these limits even after the server's overload.

Fig. 12 shows the cumulative CPU disbalance, representing the sum of CPU disbalance (in percent), based on Fig.11. It is noticeable that the LBORU mechanism gives the best results in terms of CPU balancing on servers. At the same time, Round Robin has similar characteristics since the largest part of the values (75%) is obtained in the period before the server overload.

Figures 13 and 14 show the results related to the disbalance in the upload of network resources. Namely, each response engages a different amount of these resources, which is a significant challenge for assessing each load balancing mechanism's effectiveness.

In our tests, the Random scheduling mechanism for web clients' connections gave solid CPU disbalance results. However, there is a significant disbalance observed from the aspect of network traffic. In contrast, the RR mechanism implemented on the SDN controller, despite not being aware of the links' state, still managed to ensure a satisfying network traffic balance. From the LBBSRT mechanism aspect,

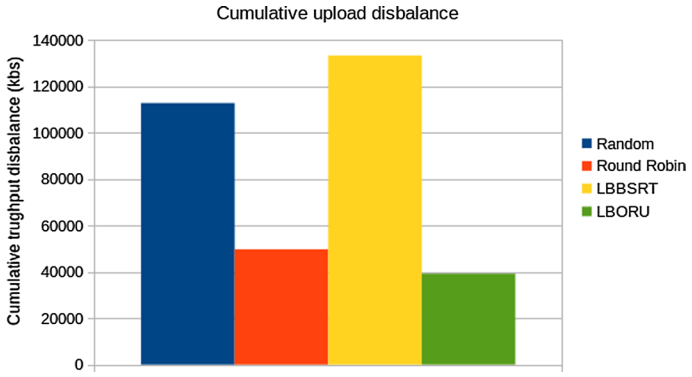

Fig. 14 Cumulative upload disbalance

we have expected better results in network traffic balancing. The reason is that the load on the network interface and the interface queue impact on RTT value increase and can indicate a disbalance, thus initiating more efficient traffic balancing on the controller. The obtained results show that this is not enough for a significant change in the RTT parameter value so that the LBBSRT mechanism can make better decisions. Tests have shown that the LBORU mechanism, which monitors the current CPU values and network resources load, makes better decisions about load balancing on servers.

7 Discussion

New technologies, the emergence of intelligent and more demanding services, new types of communications, and more numerous users demand to define a more efficient method to balance traffic. By analyzing the existing load balancing schemes described in Sect. 3, we recognized certain limitations. These schemes do not include continuous resource load detection in the decision-making process. Therefore, we have decided that the foundation of our solution is continuous detection of resource load and implementation of a mechanism that allows customization of load balancing schemes regarding service resource usage.

The obtained results indicate the influence of processing overhead on each first packet and identify the value of an initial processing delay in the SDN network. Further analysis of the relationship between the number of transactions and concurrent connections showed that the SDN concept's initial delay did not significantly affect the transaction number and average transaction time. There is a linear dependence between the number of transactions per second and the number of concurrent connections. To assess the efficiency of our load balancing scheme, we analyzed servers' resource load disbalance (primarily CPU load) as one of the efficiency indicators for load balancing mechanisms. The analysis shows that increasing the number of concurrent connections and server load makes the disbalance less noticeable.

We have discussed in the previous section the possible reasons for poor results obtained using the LBBSRT mechanism. To make it easier to compare the test results, we calculated the cumulative CPU disbalance. It is noticeable that our mechanism gives the best results in terms of CPU balancing on servers. Also, we have presented a similar disbalance analysis of the upload of network resources because it is a significant challenge in assessing each load balancing mechanism's effectiveness. The test results have shown that our mechanism, which monitors the current CPU values and network resources load (which is most significant for our test case), makes better load balancing decisions than tested mechanisms.

8 Conclusion

The rapid growth in the number of Internet users and the increasingly complex nature of their requirements impose the need for faster development of network technologies. Implementation of new services based on machine learning, IoT, neural networks, and other advanced technologies and realization of smart environments (e.g., smart city, smart traffic, smart industry) require rapid and efficient network infrastructure modernization. It is necessary to introduce a higher level of programmability and simplify network management in a heterogeneous network infrastructure. This way, we can provide the required level of flexibility and scalability in implementing existing and future services.

The SDN technology imposes itself as a logical solution in this regard. However, the complexity of full migration to SDN and the implementation costs are key challenges that impose the need to define the process of transition from the traditional to SDN network architecture in a sufficiently clear and precise way. In this sense, the incremental approach to the introduction of SDN functionality in the network infrastructure is the only economically rational and technically efficient solution today and is based on the selective implementation of SDN-aware devices and building a hybrid SDN architecture. This research shows that one can realize the hybrid SDN architecture by adding only a hybrid SDN switch and SDN controller while preserving all the traditional network's functionalities (e.g., routing, switching, DHCP, ARP, SNMP). In this way, we combine the proactive flows, which leave most of the computer network's management to traditional mechanisms, with the inserted reactive flows, used only for additional SDN functionalities.

We have evaluated our load balancing scheme in a hybrid SDN network based on the SNMP protocol for monitoring the current load of web server resources. We have applied multi-parameter metrics in the process of deciding which server to forward a connection to. This decision-making method uses many more parameters than traditional load balancing technologies and some new SDN schemes. By analyzing the obtained results, we can conclude that implementing the proposed scheme for load balancing achieves better balance in resource load and ensures efficient services implementation. Although we have tested our solution on web servers, our approach is generally applicable and can efficiently perform load balancing regardless of the traffic type.

We recognize certain limitations of our study. They are caused by how we have performed the testing according to the proposed methodology. In laboratory conditions, we have carried out the test scenario in which the testbed environment was composed of a minimum number of devices needed to realize the project objectives. This approach produces a particular limit in terms of results—they could be different if we had performed the testing in a realistic or cloud environment.

Our future research will focus on developing additional functionalities based on SDN technology, which will improve traditional computer networks' functioning and solve problems, such as IP mobility in heterogeneous networks, more efficiently. In that sense, we will direct special attention to developing new weights distribution models for coefficients, especially its influence on a large number of servers. Furthermore, part of future research will be related to a more precise identification of the impact of the proposed load balancing scheme on low latency traffic.

Acknowledgements This work was supported by the European Union's Horizon 2020 research and innovation programme under Grant Agreement Number 856967.

References

1. Sezer, S., Scott-Hayward, S., Chouhan, P., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., Rao, N.: Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Commun. Mag. **51**(7), 36–43 (2013). https://doi.org/10.1109/mcom.2013.6553676
2. Zhang, Z., Bockelman, B., Carder, D.W., Tannenbaum, T.: Lark: An effective approach for software-defined networking in high throughput computing clusters. Future Gener. Comput. Syst. **72**, 105–117 (2016). https://doi.org/10.1016/j.future.2016.03.010
3. Bojović, Ž., Bojović, P., šuh, J.: Implementing software defined networking in enterprise networks. J. Inst. Telecommun. Prof. **12**(1), 30–35 (2018). https://doi.org/10.13140/RG.2.2.10305.86887
4. Levin, D., Canini, M., Schmid, S., Schaffert, F., Feldmann A.: Panopticon: reaping the benefits of incremental SDN deployment in enterprise networks, in: USENIX Annual Technical Conference, Jun 2014, pp. 333–345. https://www.usenix.org/conference/atc14/technical-sessions/presentation/Levin. Accessed 30 Mar 2021
5. Amin, R., Reisslein, M., Shah, N.: Hybrid SDN networks: a survey of existing approaches. IEEE Commun. Surv. Tutor. **20**(4), 3259–3306 (2018). https://doi.org/10.1109/COMST.2018.2837161
6. Cardellini, V., Colajanni, M., Yu, P.S.: Dynamic load balancing on Web-server systems. IEEE Internet Comput. **3**(3), 28–39 (1999). https://doi.org/10.1109/4236.769420
7. Kaur, S., Kumar, K., Singh, J., Ghumman, N.S.: Round-robin based load balancing in software defined networking. In: 2nd International Conference on Computing for Sustainable Global Development, INDIACom 2015, pp. 2136–2139
8. Zhong, H., Fang, Y., Cui, J.: LBBSRT: An efficient SDN load balancing scheme based on server response time. Future Gener. Comput. Syst. **68**, 183–190 (2017). https://doi.org/10.1016/j.future.2016.10.001
9. POX controller (2020). https://github.com/noxrepo/pox/. Accessed 30 Mar 2021
10. Bojovic, P.D., Bojovic, Z., Bajic, D., Vojin šenk: IP session continuity in heterogeneous mobile networks using software defined networking. J. Commun. Netw. **19**(6), 563–568 (2017). https://doi.org/10.1109/JCN.2017.000096
11. Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015). https://doi.org/10.1109/JPROC.2014.2371999
12. Singh, S., Jha, R.K.: A survey on software defined networking: Architecture for next generation network. J. Netw. Syst. Manage. **25**(2), 321–374 (2016). https://doi.org/10.1007/s10922-016-9393-9
13. Peterson, L., Cascone, C., O'Connor, B., Vachuska, T., Davie, B.: Software-defined networks: a systems approach. In: Systems Approach LLC (2020)

14. Huang, S., Griffioen, J., Calvert, K.L.: Network hypervisors: enhancing SDN infrastructure. Comput. Commun. **46**, 87–96 (2014). https://doi.org/10.1016/j.comcom.2014.02.002
15. Scott-Hayward, S.: Design and deployment of secure, robust, and resilient SDN controllers. In: 1st IEEE Conference on Network Softwarization, NetSoft 2015, pp. 1–5 (2015). https://doi.org/10.1109/NETSOFT.2015.7258233
16. Lara, A., Kolasani, A., Ramamurthy, B.: Network innovation using OpenFlow: a survey. IEEE Commun. Surv. Tutor. **16**(1), 493–512 (2014). https://doi.org/10.1109/SURV.2013.081313.00105
17. Yin, H., Zou, T., Xie, H.: Defining Data Flow Paths in Software-Defined Networks with Application-Layer Traffic Optimization: U.S. Patent Application 13/915,410, 2013. https://patents.google.com/patent/US20130329601. Accessed 30 Mar 2021
18. Shin, M., Nam, K., Kim, H.: Software-defined networking (SDN): a reference architecture and open APIs. In: International Conference on ICT Convergence, ICTC 2012, pp. 360–361 (2012). https://doi.org/10.1109/ICTC.2012.6386859
19. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. ACM SIGCOMM Comput. Commun. Rev. **38**(4), 63–74 (2008). https://doi.org/10.1145/1402946.1402967
20. Messaoud, S., Bradai, A., Moula, E.: Online GMM clustering and mini-batch gradient descent based optimization for industrial IoT 4.0. IEEE Trans. Indus. Inform. **16**(2), 1427–1435 (2020). https://doi.org/10.1109/TII.2019.2945012
21. Messaoud, S., Bradai, A., Ahmed, O.B., Anh Quang, Ph.T., Atri, M., Hossain, M.S.: Deep federated Q-learning-based network slicing for industrial IoT. IEEE Trans. Ind. Inform. **17**(8), 5572–5582 (2021). https://doi.org/10.1109/TII.2020.3032165
22. Rathee, S., Sinha, Y., Haribabu, K.: A Survey: Hybrid SDN. J. Netw. Comput. Appl. **100**, 35–55 (2017). https://doi.org/10.1016/j.jnca.2017.10.003
23. Vissicchio, S., Vanbever, L., Bonaventure, O.: Opportunities and research challenges of hybrid software defined networks. ACM SIGCOMM Comput. Commun. Rev. **44**(2), 70–75 (2014). https://doi.org/10.1145/2602204.2602216
24. Vissicchio, S., Vanbever, L., Cittadini, L., Xie, G.G., Bonaventure, O.: Safe routing reconfigurations with route redistribution. In: IEEE Conference on Computer Communications, IEEE INFOCOM 2014, pp. 199–207. https://doi.org/10.1109/INFOCOM.2014.6847940
25. He, J., Song, W.: Achieving near-optimal traffic engineering in hybrid software defined networks. In: IFIP Networking Conference (IFIP Networking), 2015, pp. 1–9. https://doi.org/10.1109/IFIPNetworking.2015.7145321
26. Bahnasse, A., Louhab, F.E., Ait Oulahyane, H., Talea, M., Bakali, A.: Novel SDN architecture for smart MPLS Traffic Engineering-DiffServ Aware management. Future Gener. Comput. Syst. **87**, 115–126 (2018). https://doi.org/10.1016/j.future.2018.04.066
27. Canini, M., Feldmann, A., Levin, D., Schaffert, F., Schmid, S.: Software-defined networks: Incremental deployment with Panopticon. IEEE Comput. **47**(11), 56–60 (2014). https://doi.org/10.1109/MC.2014.330
28. Poularakis, K., Iosifidis, G., Smaragdakis, G., Tassiulas, L.: One step at a time: optimizing SDN upgrades in ISP networks. In: IEEE Conference on Computer Communications, IEEE INFOCOM 2017, pp. 1–9 (2017). https://doi.org/10.1109/INFOCOM.2017.8057136
29. Xu, H., Fan, J., Wu, J., Qiao, C., Huang, L.: Joint deployment and routing in hybrid SDNs. In: IEEE/ACM 25th International Symposium on Quality of Service, IWQoS 2017, pp. 1–10 (2017). https://doi.org/10.1109/IWQoS.2017.7969133
30. Casey, D.J., Mullins, B.E.: SDN shim: controlling legacy devices. In: IEEE 40th Conference on Local Computer Networks, LCN 2015, pp. 169–172 (2015). https://doi.org/10.1109/LCN.2015.7366298
31. Nunez-Martinez, J., Baranda, J., Mangues-Bafalluy, J.: A service-based model for the hybrid software-defined wireless mesh backhaul of small cells. In: 11th International Conference on Network and Service Management, CNSM 2015, pp. 390–393 (2015). https://doi.org/10.1109/CNSM.2015.7367388
32. Bhattacharyya, S., Katramatos, D., Yoo, S.: Why wait? Let us start computing while the data is still on the wire. Future Gener. Comput. Syst. **89**, 563–574 (2018). https://doi.org/10.1016/j.future.2018.07.024
33. Balta, M., Ozcelik, I.: A 3-stage fuzzy-decision tree model for traffic signal optimization in urban city via a SDN based VANET architecture. Future Gener. Comput. **104**, 142–158 (2020). https://doi.org/10.1016/j.future.2019.10.020
34. Neghabi, A.A., Navimipour, N.J., Hosseinzadeh, M., Rezaee, A.: Load balancing mechanisms in the software defined networks: a systematic and comprehensive review of the literature. IEEE Access **6**, 14159–14178 (2018). https://doi.org/10.1109/ACCESS.2018.2805842
35. Shang, Z., Chen, W., Ma, Q., Wu, B.: Design and implementation of server cluster dynamic load balancing based on OpenFlow. In: International Joint Conference on Awareness Science and Technology

& Ubi-Media Computing, iCAST 2013 & UMEDIA 2013, pp. 691–697 (2013). https://doi.org/10.1109/ICAwST.2013.6765526
36. Xu, Z., Huang, R., Bhuyan, L.N.: Load balancing of DNS-based distributed Web server systems with page caching. In: Tenth International Conference on Parallel and Distributed Systems, ICPADS 2004, pp. 587–594 (2004). https://doi.org/10.1109/ICPADS.2004.1316141
37. Tong, R., Zhu, X.: A load balancing strategy based on the combination of static and dynamic. In: 2nd International Workshop on Database Technology and Applications, pp. 1–4 (2010). https://doi.org/10.1109/DBTA.2010.5658951
38. Guo, Z., Su, M., Xu, Y., Duan, Z., Wang, L., Hui, S., Chao, H.J.: Improving the performance of load balancing in software-defined networks through load variance-based synchronization. Comput. Netw. **68**, 95–109 (2014). https://doi.org/10.1016/j.comnet.2013.12.004
39. Yeganeh, S.H., Tootoonchian, A., Ganjali, Y.: On scalability of software-defined networking. IEEE Commun. Mag. **51**(2), 136–141 (2013). https://doi.org/10.1109/MCOM.2013.6461198
40. Sharma, S., Singh, S., Sharma, M.: Performance analysis of load balancing algorithms. World Acad. Sci. Eng. Technol. **38**, 269–272 (2008). https://doi.org/10.5281/zenodo.1061232
41. Kaur, S., Singh, J.: Implementation of server load balancing in software defined networking. In: Information Systems Design and Intelligent Applications: Advances in Intelligent Systems and Computing, vol. 434, Springer, New Delhi (2016). https://doi.org/10.1007/978-81-322-2752-6_14
42. Song, P., Liu, Y., Liu, T., Qian, D.: Flow Stealer: lightweight load balancing by stealing flows in distributed SDN controllers. Sci. China Inf. Sci. **60**(3), 032202 (2017). https://doi.org/10.1007/s11432-016-0333-0
43. Hu, Y., Luo, T., Beaulieu, N.C., Wang, W.: An initial load-based green software defined network. Appl. Sci. **7**(5), 459 (2017). https://doi.org/10.3390/app7050459
44. Zhang, J., Xi, K., Luo, M., Chao, H. J.: Load balancing for multiple traffic matrices using SDN hybrid routing. In: IEEE 15th International Conference on High Performance Switching and Routing, HPSR 2014, pp. 44–49 (2014). https://doi.org/10.1109/HPSR.2014.6900880
45. Wang, R., Butnariu, D., Rexford, J.: OpenFlow-based server load balancing gone wild. In: 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, Hot-ICE'11 (2011). https://www.usenix.org/legacy/events/hotice11/tech/full_papers/Wang_Richard.pdf. Accessed 30 Mar 2021
46. Koerner, M., Kao, O.: Multiple service load-balancing with OpenFlow. In: IEEE 13th International Conference on High Performance Switching and Routing (2012) pp. 210–214. https://doi.org/10.1109/HPSR.2012.6260852
47. Ukrist, S., Pradittasnee, L., Kitsuwan, N.: Resolving load imbalance state for SDN by minimizing maximum load of controllers. J. Netw. Syst. Manage. **29**(4), 1–28 (2021). https://doi.org/10.1007/s10922-021-09612-w
48. El Kamel, A., Youssef, H.: Improving switch-to-controller assignment with load balancing in multi-controller software defined WAN (SD-WAN). J. Netw. Syst. Manage. **28**, 553–575 (2020). https://doi.org/10.1007/s10922-020-09523-2
49. Guo, Y., Wang, Z., Yin, X., Shi, X., and Wu, J.: Traffic engineering in SDN/OSPF hybrid network. In: International Conference on Network Protocols, 2014, pp. 563–568, https://doi.org/10.1109/ICNP.2014.90
50. Ren, C., Wang, S., Ren, J., Wang, X., Song, T., Zhang, D.: Enhancing traffic engineering performance and flow manageability in hybrid SDN. In: IEEE Global Communication Conference, IEEE GLOBECOM 2016, pp. 1–7, https://doi.org/10.1109/GLOCOM.2016.7841819
51. Lin, C., Wang, K., Deng, G.: A QoS-aware routing in SDN hybrid networks. Procedia Comput. Sci. **110**, 242–249 (2017). https://doi.org/10.1016/j.procs.2017.06.091
52. EVE-NG. https://www.eve-ng.net/ (2020). Accessed 30 Mar 2021
53. Open vSwitch. https://www.openvswitch.org/ (2020). Accessed 30 Mar 2021
54. Farhady, H., Lee, H.Y., Nakao, A.: Software-defined networking: a survey. Comput. Netw. **81**, 79–95 (2015). https://doi.org/10.1016/j.comnet.2015.02.014
55. Bojovic, P. D., Malbasic, T.: SDN-LBORU—load balancing by optimizing resource utilization. GitHub repository. https://github.com/Paxy/SDN-LBORU (2020). Accessed 30 Mar 2021

Teodor Malbašić is a Ph.D. student at the University of Novi Sad, Serbia, where he received a master's degree at the Faculty of Technical Sciences. He currently works as a software engineer at AUTOSOFT LLC in Novi Sad. His research interests include computer networks, Big data, and IoT.

Petar D. Bojović is an Associate Professor at the School of Computing Union University in Belgrade. As the researcher and teacher, he is covering the courses in computer networks and network security. He received the Ph.D. degree in Computer Engineering from the University of Novi Sad in 2019. Currently, he works in the area of computer network security and software-defined networking.

Živko Bojović is an associate professor at the University of Novi Sad. He received a Ph.D. at the Faculty of Technical Sciences in Novi Sad. Bojović is a member of the Centre for Intelligent Communications and the IEEE, and his research interests include computer networks, e-learning, Big data, and IoT.

Jelena Šuh is an engineer for technology strategy in "Telekom Srbija". She received a Diploma degree from the Faculty of Electrical Engineering, University of Belgrade, Serbia and a Ph.D. from the Faculty of Organizational Sciences, University of Belgrade, Serbia. Her research interests include computer networks, Internet technologies and e-education.

Dušan Vujošević is an associate professor at the Union University School of Computing in Belgrade. His research and teaching interests include decision support systems, integrated information systems, and human–computer interaction. Working on many commercial and scientific projects, he has got the experience with project management and business intelligence technologies.