IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 21 November 2023, accepted 2 January 2024, date of publication 10 January 2024,
date of current version 19 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3352433

RESEARCH ARTICLE

Impact of SDN Controller's Performance on Quality of Service

SYLWESTER KACZMAREK AND JACEK ANDRZEJ LITKA
Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, 80-233 Gdańsk, Poland
Corresponding author: Sylwester Kaczmarek (kasyl@eti.pg.edu.pl)

ABSTRACT Software Defined Networking is a paradigm in network architecture; that is quickly becoming commonplace in modern telecommunication systems. It facilitates network customization for the requirements of different applications and simplifies the implementation of new services. Since its proposal, a significant evolution in its functionality has occurred. However, this development brought along problems of efficiency and performance, which are currently under research. A number of requirements has to be met, if Software Defined Networking is going to be the next step in the Next-Generation Networks progression. The central part of it – the SDN controller – has to put minimum strain on the system and provide performance which does not impede Quality of Service requirements. In this paper, the results of a research on SDN controller's performance have been provided in the context of keeping up with flows' QoS. For this, an emulated-physical research platform has been implemented. This research environment utilizes traffic generated accordingly to ITU-T recommendations to validate QoS parameters. The platform is given a thorough description. The results obtained from it take under consideration the implemented traffic sources, as well as the intensity of traffic handled by the controller and the traffic load of data plane links. Authors indicate that even without breaking the limitations set for delays in QoS, the impact of the controller workload is noticeable, which should be mitigated by applying appropriate resource control.

INDEX TERMS Emulation, performance evaluation, quality of service, software defined networking.

I. INTRODUCTION

Software Defined Networking (SDN) is an approach to the management and operation of resources and teletraffic, that utilizes Next-Generation Networks' (NGN) separation of control and data planes. This is performed by moving the logic of traffic control away from the nodes to a separate entity known in SDN nomenclature as the controller [1]. This entity is responsible for deciding on the rules for traffic control and it may simply be restricted to already established well-known routing algorithms. However, with more sophisticated implementations, more complex set of rules taking under consideration some nuances – like the type of the traffic in the flow – may be utilized to control how traffic is forwarded in the network.

This is a significant part of the functionality research conducted within the SDN architecture. A number of papers published in recent years tend to focus on these aspects of the technology: presenting optimized rulesets for controlling the traffic and novel applications working with the controller for this purpose, or use cases on utilization of this architecture in modern and future networks.

The SDN architecture is generally described as consisting of three planes. The bottom one is the Switch Plane, which includes nodes (switches) responsible for forwarding the traffic. However, they do not take active part in designing the rules of forwarding. The middle plane is the Controller Plane, which includes controllers connected to the switches. They are responsible for managing the Switch Plane and decide upon the rules of traffic forwarding, which they install in the nodes. The upper plane is the Application Plane, which consists of applications connected to the controller. They work with advanced algorithms and sets of rules, allowing for more sophisticated traffic forwarding. The results of their work are provided to the controllers, which translate them into sets of rules to be installed in the switches.

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

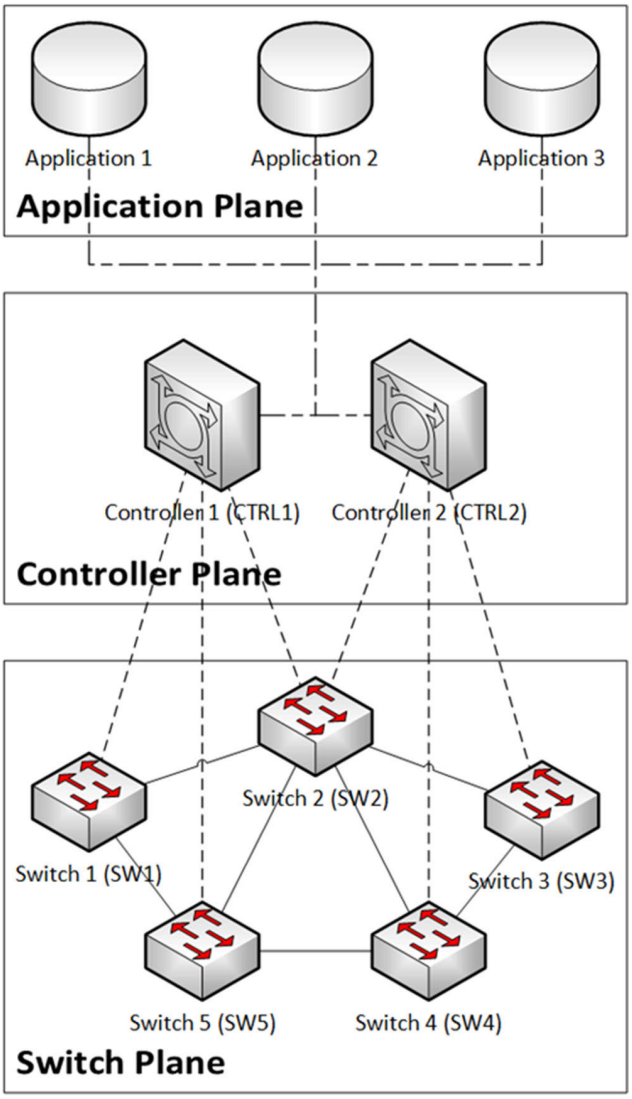

FIGURE 1. Generalization of SDN architecture.

An illustration of a generalized SDN architecture is presented in Fig. 1.

SDN paradigm has gain popularity and is becoming a commonplace implementation for datacenters and enterprise networks. No wonder then that with promising results from the functionality aspect sphere of research, a push for deployment in other types of networks – like in operator networks – has emerged in recent years.

This defines some requirements for the implementation, most of which are concerned with performance of the architecture. Authors of an extensive survey on SDN [1] have already noted that the performance aspect has not been clearly defined during the SDN architecture design, which opens the door for research in this field. Of course, the engineering and scientific communities have already taken part in this process, giving us many papers [2] and proposing tools, like cbench and Mininet.

This problem becomes even more relevant with different types of traffic serviced by data plane imposing different requirements on the network. These requirements, known collectively as Quality of Service (QoS), set restrictions on the maximum values of ITU-T defined parameters. Among these QoS parameters there are three crucial ones: IP Packet Transfer Delay (IPTD), IP Packet Delay Variation (IPDV) and IP Packet Loss Rate (IPLR) [3].

SDN networks are known for their ability to easily adapt to the changing nature of the teletraffic. These changes require a high level of performance from the controller. The controller itself needs to be adjusted for efficient work in such an environment, as its working lead to additional delay added to the time it takes to forward a packet through the Switch Plane. This is essential, since the QoS requirements for certain types of teletraffic may be significantly more restrictive, which could lead to additional delay making the network's architecture not compliant with them.

Therefore, it is necessary to define what exactly SDN performance is. The authors decided to distill the problem into the subject of controller's performance. The time of response was proposed as a measure of the controller's performance. To be suitable for SDN utilization, this parameter has to be as short as possible, even with an increase of controller's workload. The time of response of the controller translates directly into the delay it imposes on the handled flow, so by regulating the intensity of the controller's workload and observing its time of response, we can evaluate its performance. An efficient controller should have such a low time of response, that the delay it adds to the flow should not lead to breaching the maximum allowed delay for the flow's QoS.

This paper covers research conducted to highlight the possible impact the controller's performance has on keeping up with the QoS requirements. A formal model for SDN network is presented, which has been implemented with the use of a research platform consisting of both emulated and physical entities. A series of measurements utilizing emulated real-life traffic types has been conducted in it, in order to indicate the influence of the controller's performance on the packets' end-to-end delay. The results have been analyzed and confronted with recommendations of the ITU-T.

The novelty of authors' research lies in implementing a hybrid emulated-physical framework and utilizing real-life traffic. With this, a testbed for realistic use cases for SDN controller's performance in operator networks has been designed. Analysis conducted in it provide insight into the influence of controller's performance on the end-to-end delay of forwarded traffic, as well as its impact on the distribution of delay values, and how does it comply with the required QoS.

The reminder of this paper is organized as follows. Section II gives greater detail on the performance problem in SDN networks. Section III presents the research platform used in this study. In Section IV, the results of the experiments are presented. The final conclusion is available in Section V.

II. PROBLEM OF PERFORMANCE IN SDN NETWORKS

In this section, an overview of the problem of SDN networks' performance (mainly focused on the controller) is provided. This outlines the background for the conducted research. Conclusions regarding the necessary adjustments for the research community are provided at the end of it.

In case of SDN networks, numerous researches have been conducted to define the functionality range of the architecture. Aside from that, scientific community works on novel algorithms and optimization approaches leading to more reliant networks which follow the software defined principle.

With the advent of programmability and the application-based control over the network, a field for research into the performance of the network emerges.

The performance in the case of telecommunication networks can be defined as the ability of the network to keep up with the workload generated by the traffic traversing it. In case of the SDN, this can be narrowed down to the ability of the controller to keep up with the demands (thanks to its time of response), as has been noted in the previous section. This means, that an efficiently performing network should be capable of guaranteeing successful access to the services, as well as their positive results. In this case, an ability to send and receive information in a manner compliant to the time requirements – like having low and constant delay values during a phone connection – and guaranteeing little to no loss of information. Taking under consideration that in a typical data center there may be up to 200,000 flows per second incoming, problem of performance becomes essential [4]. In SDN's case this can be separated into number of aspects, some of which are already being covered by the scientific community.

The research has already taken under consideration a range of topics, like the time of response of a controller [5], [6], the number of flows the controller is capable of managing in a period of time [7], the response time of the switches [8], the overall delay in SDN-domain based network [9], controller's placement in the network [10], intensity of traffic required for setup and upkeep procedures [11], [12], and more. Mostly, a constant bit rate (CBR) types of traffic are utilized during the experiments in the testbeds.

Some solutions for decreasing the time of the controller's response were also provided, like project DevoFlow [13] and DIFANE [14], they however require modifications of the switches and not the controller, which in a way misses the principle of SDN network. This being the centralized controller's objective to handle the process of designing the logic behind traffic forwarding [15].

In modern telecommunication networks, the kind of traffic carried is diverse and depends on the kind of information encapsulated in it. CBR is not enough to appropriately relay the true nature of teletraffic, as many sources generate flows of variable bit rate (VBR). After all, traffic model for voice service will differ from data and video [16]. Because of that, measurement for the number of flows a controller (or a cluster of them) can handle and the delay it imposes on the data plane is not enough. The results of research on controller's capacity may complement a uniform traffic distribution, but not – for instance – an ON/OFF traffic model, which is akin to how the traffic for telephone VoIP connections look like [16]. The characteristics of the flow differ, depending on the source. It is such an important aspect, that standardization work has already been done by ITU-T for the purpose of defining the parameters of the flows used for network performance tests in regards to keeping up with QoS [16].

The authors mention it, because of many services' VBR nature, the information on how many flows a controller is capable of handling is not enough for assessing the architecture's level of performance. Even if the number of handled flows is an enormous number, the controller will impose some sort of delay on them. It might be compensated, when it comes to CBR flows. For VBR packet streams – which current bitrate differs depending on the time window of the observation – compensating for the delay is not a trivial hurdle to overcome.

This means, that the analysis of the controller's performance is highly dependable on the traffic characteristics of the flows it handles.

The types of traffic existing in the modern network are a vast collection. Traffic generated by user services will diverse in shape. This is nothing new and has been a subject of discussion for a long time. Packet switching network does not guarantee QoS by default, and has to be improved by different means of guaranteeing it.

Some works in this regard have already been done. With the implementation of Type of Service (TOS) field in IP packet it has become feasible to sort out which traffic is more demanding in resources, in order to guarantee better quality of provided service. What is more, the OpenFlow protocol (the most common protocol utilized in SDN networks) supports this value, which means that an OpenFlow-enabled switch can match flows by it. This gives the network an ability to handle such traffic accordingly, so that QoS might be supported.

It does not implement QoS assurance mechanisms in the SDN architecture itself. Even though SDN switch might recognize a high priority flow (by matching the TOS value), the controller stays priority-agnostic. Without any kind of external augmentations, the default SDN architecture is incapable of prioritizing some traffic over the other. For instance, by default OpenFlow does not guarantee any kind of algorithm that would allow the controller to recognize which packet-in messages – which are the messages informing the controller that a flow has to be handled, when there is no matching rule in the switch – are originating from traffic with a higher priority. This leads to unwelcome results. These phenomena have to be addressed, as different kind of traffic might be influenced differently than the others. Especially when taking under consideration, that each traffic type can be treated as a different kind of generator, as described in ITU-T recommendation [16].

With the information mentioned in the previous paragraphs, an image of the negative influence of the controller on the delay arises. The additional delay comes from the fact, that when handling a stream by the controller (by which we mean analyzing every packet-in action), it has to check the packet of the flow, to make sure whether it should be forwarded in the network and what are the necessary actions to be performed by its nodes. This takes time, as the controller confronts the packet with own base of possible flow rules. When the controller's reaction is too slow, we experience events, in which packets from different flows start to gather in a queue, awaiting handling. Each waiting packet increases its delay and too long of a handling time leads to situation in which every next flow's delay increases (due to its waiting for being handled). Not only the awaiting new flows, but also the ones already being handled, experience additional delay. This happens when packets of the current flow are being handled slower than the time of arrival of the next packets of that flow. Controller not differentiating between the priority of flows, leads to situations in which a high QoS-requirement flow has its delay increased, simply due to the fact, that the controller is still handling earlier requests, some of which might come from lower priority flows.

Different types of traffic and their influence on the SDN network have piqued scientific interest, leading to quite interesting findings and observations like the analysis of flow table occupancy [17], or the focus on the impact of the nodes (switches) resources and bandwidth [18].

SDN controller is the central part of this network architecture. Which means, that it's performance will have a tremendous impact on the efficiency of the network. The goal of the controller is to observe the network structure, react to changes and decide upon rules, according to which the different flows are forwarded by the nodes.

Both the proactive and the reactive types of controllers acting modes will require a great performance from the hardware and software on which they relay. If the SDN network has to be applied in an environment with a big number of traffic sources that rapidly change their target destination or even the characteristics of their generated flows, then a great deal of effort has to be put into adjusting the controller in such a production environment. The number of requests for packet handling would be huge. This would be an especially valid concern in cases where the different flows cannot be that easily aggregated, each of them requiring a separate set of forwarding rules. Such use cases become more common with the ever-increasing usage of Internet of Things (IoT) services and appliances in the network.

A proactive controller has to take this into account and react quickly to the ever-changing intensity of traversing flows. Reacting with haste to the necessities arising, even before them being signalized, is a troublesome hurdle requiring specialistic algorithms and a great deal of performance boost to not influence negatively the time requirements set by the flows.

In case of reactive mode, when a new flow emerges, it is accompanied by the requirement of installing a flow rule in the switches' tables. In such cases a switch sends over to the controller a request for action, which results in either dropping the packet, or installing a set of rules, according to which the flow will be forwarded. With a truly dynamic network, which is traversed by a great deal of different packet flows, each new additions procure a request for handling. This requires from the controller's performance to be on par with the great influx of requests generated by the network. The controller which handling of the requests is simply too slow will impact the network in a negative manner leading to the increase of packet delays and changing the variance of it to undesirable levels.

To include the effects of the controller's performance on the network and it's keeping up with the QoS requirements, research has to be conducted which includes real-life-like traffic in the experimental framework. Such research sheds light on what is leading to negative impact on the network and outlines the necessary changes into its implementation, or the architecture as a whole.

By understanding the nature of the problem outlined in this section, the authors have decided to conduct research, which takes the aforementioned doubts into account. A research environment for SDN performance – no matter if it focuses on the controller, or on the nodes – should utilize traffic no different from real-life use cases. Even when most of the different flows are aggregated for the simplification of its forwarding, we have to take under consideration, that with new emerging services, there exist events of sudden bursts of short living, yet high priority flows. This means, an SDN controller's performance has an incredibly high role in providing the necessary level of service handling, to make sure the flows comply with the QoS.

An environment for such experiments to be conducted – with focus on a single controller network – is presented in the following section of this paper. In the later sections authors provide the results of research, that was conducted in the described testbed.

III. CONTROLLER'S PERFORMANCE RESEARCH PLATFORM

The hybrid approach of the platform required preparing a number of computer machines. The authors decided to utilize three PC machines, all of which working under the control of Linux OS (specifically speaking, Ubuntu 18.04). One of the PCs was working as the single controller, the second as the emulated domain, and the last one as the generator of the real-life-like traffic.

Both the controller's, as well as generator's PCs, tech specs were: Intel Core i7-4770 CPU (3.4 GHz, 8 cores), 16 GB of RAM and a Gigabit Ethernet NIC. The last component of our framework was the emulator, which run on a single PC machine. The tech specs for this computer were: AMD Ryzen Threadripper 1920X CPU (3,5 GHz, 24 cores), 64 GB of RAM and two Gigabit Ethernet NICs.

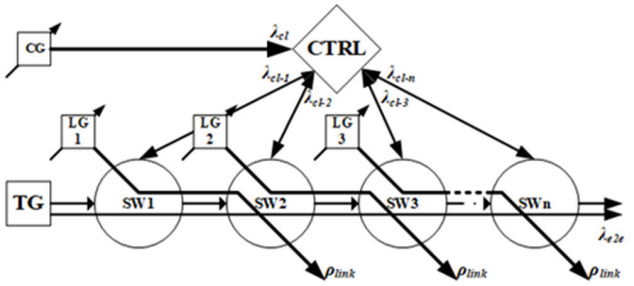

FIGURE 2. Proposed SDN performance model.

A. MODEL'S FORMAL DESCRIPTION

For the research purposes authors have come up with a formal description of the model. The model consists of $n+1$ entities, which include a single controller (CTRL) and n number of switches (SWn). Each switch generates traffic forwarded to the controller, which consists of packet-in messages the controller has to handle (this was mentioned in previous sections of this paper). After handling the packet-in message, the controller generates a flow-mod message, which includes the rules of how to forward the packets of the flow from which the packet-in message came. Alongside the flow-mod message, the original packet is being returned to the network, encapsulated in the packet-out message. The intensity of this traffic is denoted by λ_{cl-i}, where i is the index of the switch. In the model authors decided on a chain-like structure of inter-switch connections, which means, that the index i equal to 1 will mean the first switch in the chain (an entry point for the teletraffic) and index i equal to n will denote the last switch of the domain (an exit point for the teletraffic). Traffic λ_{cl-i} is highly dependable on the number of flows, which the teletraffic consists of and the time of their flow rule expiration in the i switch. In previous paragraph the term "teletraffic" is used, by which authors mean the traffic generated by their generator and working as an emulation of real-life traffic traversing telecommunication networks. This is the end-to-end traffic, which delay is used to validate the performance of the architecture. From this point it will be denoted by λ_{e2e}. The generator in author's testbed is called Traffic Generator (TG).

To measure the performance of the controller, extra traffic handled by it, called load traffic λ_{cl}, was added. This traffic was generated with the use of Controller Workload Generator (CG), which targets the single controller utilized in this architecture.

In this performance model one more parameter is included, which is the background traffic. The background teletraffic is denoted with ρ_{link} and is generated with the use of Link Workload Generators (LGs).

The proposed model for SDN controller's performance is illustrated in Fig. 2.

The parameters that are regulated are λ_{cl} (from CG) and ρ_{link} (from LGs). λ_{e2e} (from TG) and λ_{cl-i} (from SWs) are parameters that randomly change in each run of the experiment, the reasoning behind it is explained in Section IV.

Before the implementation, the authors have conducted an analysis of the network behavior of the SDN architecture. The goal was to propose the metrics for performance evaluation and decide upon the requirements put on the architecture that had to be fulfilled to guarantee a desired level of performance.

The times in the architecture have been described as follows: the total time of the packet traversing the SDN domain (t_{e2e}), consist of both the time in the controller (t_c) and the time spent in the switches (t_s). By "time spent in the switches" authors mean not only the time in the nodes themselves, but also the time spent in the links in-between them.

Time in the controller consist of five components: time of transmitting to the controller (dependent on the interface transmission speed) (t_{ts2c}), time of propagation on the switch to controller link (t_{ps2c}), time of handling by the controller (t_{ch}), time of transmitting the packet back to the switch (t_{tc2s}) and the time of propagation on the controller-to-switch link (t_{pc2s}). For the n number of switches in the domain the mean value of t_c is (1).

$$E\left(t_c\right) = E\left(t_{ts2c} + t_{ps2c} + t_{ch} + t_{tc2s} + t_{pc2s}\right) \tag{1}$$

Times of propagation is dependent on two components. One being the length of the link between switch and the controller (s_{c2s}) and vice-versa (s_{s2c}), the other being the delay per kilometer for the links (t_{link}), which is defined by the velocity of propagation of the signal in the transmission medium. The lengths are typically presented in kilometers and the delay of the link in case of an optical link is 5 μs per kilometer. This means that (1) can be presented as (2).

$$E\left(t_c\right) = t_{link} \cdot \left(s_{s2c} + s_{c2s}\right) + E\left(t_{ch} + t_{ts2c} + t_{tc2s}\right) \tag{2}$$

What has to be remembered at this point, is the fact, that controllers might work in a different way when handling the packet. They can either install the flow rule just for the switch from which the packet came, or send flow rule installation messages to all switches on the path of the given flow.

When it comes to the time in the switches, authors distinguish three components: time of handling by the switch (t_{sh}) (which includes the work of the switch, aside from waiting for the flow rule to arrive), time of transmitting (t_{st}) and time of propagation between the switches ($t_{i,i+1}$). The i value is from the range $<1; n>$, which includes the order of the switches in the chain.

The mean t_s formula for a domain with n switches is (3).

$$E\left(t_s\right) = \sum_{i=1}^{n} E\left(t_{sh_i} + t_{st_i}\right) + \sum_{i=1}^{n-1} E(t_{i,i+1}) \tag{3}$$

Time of propagation is dependent on the length of the link in kilometers ($s_{i,i+1}$) and the delay per kilometer. The links are again fiber optic lines. With this in mind, we can transform (3) into (4).

$$E\left(t_s\right) = \sum_{i=1}^{n} E\left(t_{sh_i} + t_{st_i}\right) + t_{link} \cdot \sum_{i=1}^{n-1} s_{i,i+1} \tag{4}$$

The formula of mean t_{e2e} for packets of the flow sent to the controller is (5).

$$E\left(t_{e2e}\right) = E\left(t_c\right) + E\left(t_s\right) \tag{5}$$

Depending on how the first packet in the domain is treated the (5) formula can be presented as either (6) or (7). Equation (6) is for cases when a packet of the flow is forwarded to the controller only once and the controller installs the flow rules for all the switches in its path.

$$E\left(t_{e2e}\right) = E\left(t_{ch} + t_{t_{s2c}} + t_{t_{c2s}}\right) + \sum_{i=1}^{n} E\left(t_{sh_i} + t_{st_i}\right) + t_{link} \cdot \left(s_{s2c} + s_{c2s} + \sum_{i=1}^{n-1} s_{i,i+1}\right) \tag{6}$$

Equation (7) is for cases when a packet of the controller has to be sent to the controller, when entering each switch in the path. In these cases, the t_c will be repeated n times.

$$E\left(t_{e2e}\right) = t_{link} \cdot \left(\sum_{i=1}^{n}\left(s_{s2c_i} + s_{c2s_i}\right) + \sum_{i=1}^{n-1} s_{i,i+1}\right) + \sum_{i=1}^{n} E(t_{ch_i} + t_{t_{s2ci}} + t_{t_{c2si}} + t_{sh_i} + t_{st_i}) \tag{7}$$

With all of the above formulas in mind, a mathematical descriptor for the correct work of the SDN architecture has been established.

In the case of this particular model, the authors decided to utilize the (6) formula variant, as they consider it to be a more viable solution.

Knowing the times in the domain, authors decided on the metrics to be used for the validation. This being the ratio of the mean time in the controller to the mean end-to-end time (8) and the ratio of mean time in the controller to the mean time in the switches (9).

$$m_{c/e2e} = \frac{E\left(t_c\right)}{E\left(t_{e2e}\right)} = \frac{E\left(t_c\right)}{E(t_c) + E(t_s)} \tag{8}$$

$$m_{c/s} = \frac{E\left(t_c\right)}{E\left(t_s\right)} = m_{c/e2e} \cdot \frac{E\left(t_{e2e}\right)}{E\left(t_s\right)} \tag{9}$$

For the domain to work correctly, the flow-message for a given flow has to reach a switch before the next packet of that flow. When a first packet of a flow reaches the first switch in the chain, it triggers an event of preparing a new flow rule to be installed for the rest of the switches in that path (typically, by encapsulating that packet, sending it over to the controller and then returning it back to the network with corresponding flow-mod message). This means that for the network to work correctly, the second and each next switch in the path should receive a new flow rule to install, before they receive a next packet of that flow.

The i value includes the index number of switch in the path, excluding the first switch, which ends with n equal to the total number of switches in the path. For each of the i, (10) has to be fulfilled:

$$t_{arr_{pi}} < t_{arr_{fmodi}}, \tag{10}$$

where t_{arrpi} is the time of the arrival of the packet at the i switch and $t_{arrfmodi}$ is the time of arrival of the flow-mod message to the i switch from the controller.

The t_{arr} is equal to the sum of t_s and t_c for all the switches before the switch i.

The situation in the case of the first switch in the chain (SW1) for the beginning of the flow is illustrated with Fig. 3.

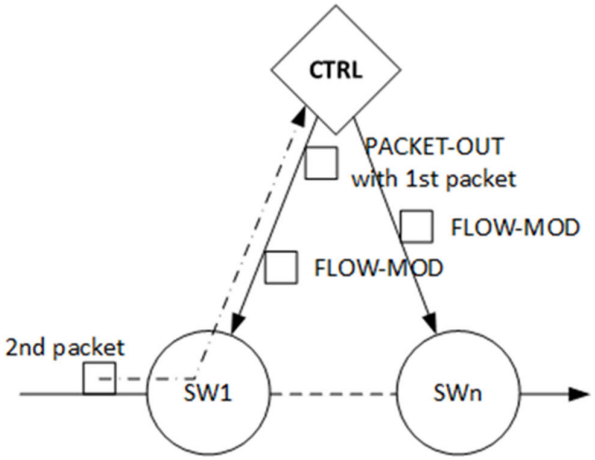

FIGURE 3. Current of events in the SDN domain. Second packet of the flow has to arrive after the rule from the FLOW-MOD packet has been installed.

The $t_{arrfmodi}$ denotes the time it takes the rules of forwarding the flow to reach a switch. The value for an i switch can be described with the equation (11).

$$E\left(t_{arr_{fmodi}}\right) = E\left(t_{t_{c2si}}\right) + t_{p_{c2si}}, \tag{11}$$

where t_{tc2si} is the time of transmitting a packet from the controller and t_{pc2si} is the time of propagation of the packet to the switch. Both of these values are taken from the t_c formula.

The mean value of t_{arrpi} can be calculated from the formula (12).

$$E\left(t_{arr_{pi}}\right) = E\left(t_{arr_{fmod1}} + \sum_{j=1}^{i-1} E\left(t_{sh_j} + t_{st_j} + t_{j,j+1}\right)\right) \tag{12}$$

Note the presence of t_{cfmod1} for the first switch. It is there, as the packet entering the domain will not continue its path, until it is sent further by the first switch. As we have discussed earlier, the model works under the principle that the first packet of the flow triggers the controller to install the rules for its forwarding in all of the switches in its path.

The t_{arri} can be further presented as (13).

$$E\left(t_{arr_i}\right) = E\left(t_{t_{c2s1}}\right) + t_{p_{c2s1}} + \sum_{j=1}^{i-1}\left(E\left(t_{sh_j} + t_{st_j}\right) + t_{j,j+1}\right) \tag{13}$$

With both (11) and (13) substituted in (10) we conclude with the final (14).

$$E\left(t_{tc2s1}\right) + t_{pc2s1} + \sum_{j=1}^{i-1}\left(E\left(t_{sh_j} + t_{st_j}\right) + t_{j,j+1}\right) < E\left(t_{tc2si}\right) + t_{pc2si} \tag{14}$$

By fulfilling the requirement (14) we design a domain that works correctly. Without it, events occur in which the packet reaches the switch before the rule for it is installed, which would in turn trigger sending the packet as packet-in message to the controller, having to handle it and install the rule again, even though it has already been sent beforehand. This increases the workload for the controller. The more it happens, the bigger delay it imposes on the network.

B. REQUIREMENTS AND CONCEPT FOR THE RESEARCH PLATFORM

The overall concept behind the research platform required to design a framework, which implements an environment in which teletraffic akin to realistic case scenarios could be used. It was also crucial to allow regulation of parameters, which would lead to ability of changing them, so that observation of their impact on the performance of the network could be conducted. Measurements should be done during the period of experiments' runs and be stored in a format which can be easily transferred to an external entity (either application or a device) for analysis. The general form of the proposed testbed is presented in Fig. 4.

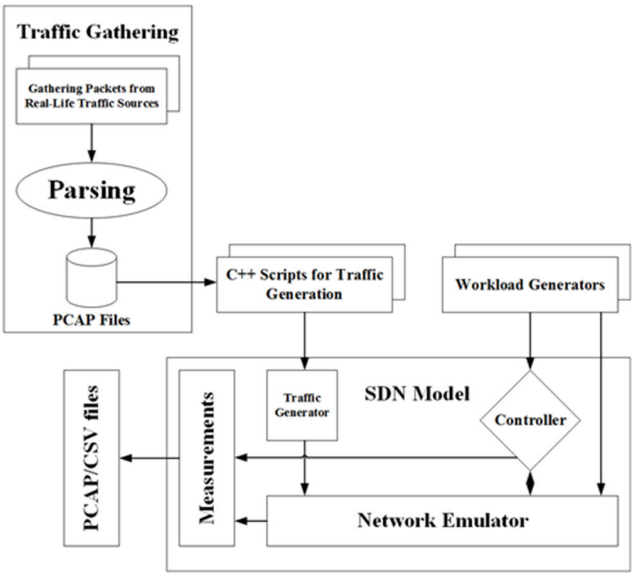

FIGURE 4. Testbed for SDN controller's performance.

SDN Model block includes switches SWs implemented in Network Emulator and traffic generator TG. Workload Generators block implement CG and LGs.

The framework produces workload for controller and emulated network from generators, which are implemented with the use of MGEN traffic generator [19]. These generators are launched from the level of custom C++ scripts, which define the pattern of packets generation from PCAP files [20], that were gathered earlier from real-life IP traffic.

The results are provided in PCAP and CSV files, which can later be exported to external tools for analysis and calculations.

The details on the structure of the network utilized in the model are provided in the following subsection.

1) NETWORK STRUCTURE

Design of the structure was created with a specific scenario in mind. The topology emulates a bottleneck from a real-life telecommunication network and adjusts it to an SDN architecture philosophy. The structure consists of a number of switches (SWs), a single controller (CTRL) and generators. It is illustrated in Fig. 5.

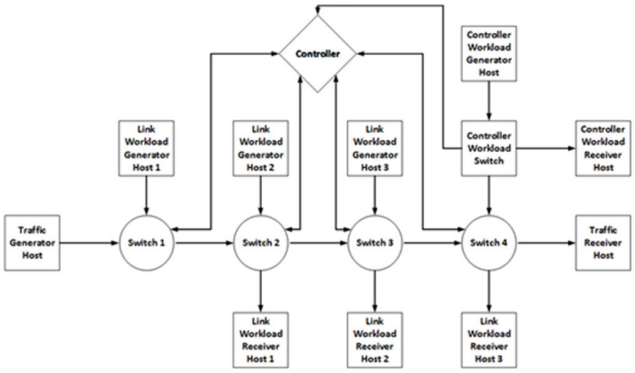

FIGURE 5. Network structure in the proposed testbed.

This particular structure consists of four SDN-compatible switches (SW1-4), a single SDN controller (CTRL), link workload generators (LG1-3) and their receivers (three pairs in total), a controller workload generator (CG) and its receiver and a traffic generator.

The specific goal of the controller is presented in the later subsection of this paper. For now, it will suffice to describe it as a central element of the structure, tasked with management of flow rules.

The switches are the network's nodes of data plane. Their task was to forward traffic traversing the network according to flow rules, which have been decided by and installed by the controller. All of them were connected to a controller via separate point-to-point network. The switches were available under a single IP address. Each of them opened a unique TCP session with the controller, making it possible to recognize the switches by TCP ports. Such a connection is called out-of-band (or simply out-band) and allows separation of resources for traffic forwarding and control.

The model incorporates both physical and emulated approach dividing the elements of its structure between them. Controller and traffic generator have been implemented as physical, distinct machines; the rest has been implemented in emulated environment with the use of Mininet emulator [21].

Mininet works by launching a number of virtualized Open vSwitches (OVS), hosts, links between them and even virtual controllers in Linux user space. Switches and hosts are visible as self-contained virtual machines, allowing the host machine's operator to launch host's applications in any of these virtual machines, simplifying the benchmarking and versatility of the experimental environment. Additionally, Mininet allows the VMs to connect with physical devices outside of the host machine, which include switches, hosts and controllers, which was an important factor in choosing this solution. Currently, Mininet is an industry standard for research on the SDN networking subjects, utilized in a number of research papers referred by the authors.

This structure allows expansion in case of such necessity arising. A chain-like connection of switches can be lengthened by adding additional nodes to it.

The emulated nature of the model allows for easy scalability. Implementation of the controller and traffic generator as physical machines mitigates the necessity to share the resources between them and the emulated domain, making sure that the controller and generator do not impact the performance of the domain itself. It is important, as the generation of traffic that is not uniform in nature is a complex and highly resource dependent task. What is more, it makes the project vendor-agnostic, allowing any kind of an OpenFlow-enabled SDN controller to be connected to the domain.

Generators used in the structure followed different principles, all of which are described in the next subsection.

The emulator was connected to a controller via single NIC card. The second NIC card was used to connect with the generator. Description of the traffic sources is provided in the following subsection.

2) TRAFFIC SOURCES

There are five traffic sources in this testbed. Three of them generate traffic which is tasked with working as workload for the links themselves. Regulating the intensity of them allows to observe the influence of link load on the performance of SDN network. One generator produces traffic which is send via Controller Workload Switch to generate workload for the controller. These four generators are emulated alongside the teletraffic in domain. The last generator is a physical machine connected to it, and is tasked with generating the traffic that emulates real teletraffic and as such differs from the previous four in a pronounced way, which will be detailed later in this section.

Link workload generators generate traffic in a uniform manner, with constant number of same-sized packets per second. They produce flows in different relations. All of these flows are CBR in nature and require simple forwarding rules in the switches' flow tables. In a manner of fact, this traffic should not produce packet-in messages, aside from a single one at the very beginning of traffic generation, as the rules that manage them should not be removed. Their single objective was to create workload for the links themselves. The workload is devised from a simple equation (15).

$$\rho_{link} = \frac{\alpha \cdot \lambda_{link}}{c} \tag{15}$$

where c is the link bandwidth (in this case 1 Gb/s), α is the size of the packet and λ_{link} is the intensity of traffic generated by the Link Workload Generator. The value ρ_{link} (link load) is one of the parameters we regulate in our model allowing it to reach values from the range $<0; 1>$.

Controller workload generator generates traffic in a uniform pattern as well. This workload does not weigh on the links in-between the switches, as they are directly sent to the controller, to generate packet-in events. The details on this process are provided later in this section, in its final subsection. The packets of this workload are of the same-size. By changing the number of same-sized packets a flow sends per second, we regulate the intensity of the controller's workload (λ_{cl}).

The last generator (''traffic generator'') is a physical machine connected by Gigabit Ethernet to the emulated domain. This network entity is the origin point of the traffic, which emulates real-life examples and is utilized to verify whether the architecture keeps up with completing the QoS requirements. Because of that, authors have decided to diversify it by guaranteeing different types of traffic to originate from that single generator.

The diverse number of traffic types generated by author's generator conforms to the existing teletraffic in networks. Most works done on the controller's performance do not take that under consideration in their performance assessments [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35].

This stream of traffic consisted of n_{flows} different flows, which were divided into three separate groups, each of which had characteristics of a different service-type stream: voice (VoIP), video (VOD) and data (FTP). This traffic generator is a single physical generator, but consisting of a number of software generators, cooperating to produce the teletraffic for the domain.

The voice flows were most numerous, consisting of a total of n_{VoIP} different flows. Each of the flows had a maximum bit rate of 64 kb/s. Each of them was implemented as a different software generator, with ON/OFF characteristics, as presented in [16]. The parameters of the generator were based on [36], which is an ITU-T recommendation that defines experimentally a model for an artificial representation of a phone conversation in English. In [36] the talk-sprout lasts on average for about 1 s, in case of pause the average is about 1.5 s. For this research, it has been decided to correspond this into our voice-traffic generators, treating a talk sprout as an ON period and pause as an OFF period.

There was a total of n_{VOD} video flows. Each of them was created by replicating an appropriate PCAP file [20]. These files have been generated by capturing 30-minutes long traffic from a point-to-point Gigabit Ethernet connection between two computers during an RTSP video transmission. The

video transmitted was encoded with MPEG-4. Ten different recordings have been utilized, each providing a separate file. The video traffic does fulfill self-similarity criteria. During validation it was observed that the Hurst parameter for all of these recordings did range from 0.6957 to 0.814, making them flows compliant with the ITU-T recommendation [16].

The last n_{FTP} flows were data-oriented. They have been created by replicating PCAP files, as in the case of video flows. These files were generated by capturing traffic originating from an FTP download session between two computers. These sessions were limited to have a maximum available bandwidth of about 10 Mb/s (by limiting the download on the FTP server side). The sessions differed by the size of the file downloaded by the client. The available file sizes were: 1 GB, 350 MB, 67 MB and 8 MB. By guaranteeing different file sizes, we have provided four different flows with different traffic characteristics.

In all of the presented in this subsection cases, for traffic generation the MGEN packet generator [19] was utilized. Details on the implementation are presented in the fourth subsection. The complete set of teletraffic is summarized in Tab. 1. In the table the bandwidths for each flow types have been presented.

TABLE 1. Types of traffic utilized in the paper.

Traffic Type	Bandwidth of a Single Flow
Voice (VoIP)	≈ 64 kb/s
Video on Demand (VOD)	<1.985; 5.656> Mb/s
File Transfer (FTP)	≈ 10 Mb/s

After defining the types of traffic traversing the domain, the authors have defined the objectives of the controller. This has been provided in third subsection.

3) CONTROLLER'S OBJECTIVES

SDN controller is the central entity in the architecture of Software Defined Networks [1]. Its main goal is to provide rules for traffic forwarding in the network. However, it is not the sole task of the controller, as it is also capable of gathering information on the network, e.g., utilization metrics or the data on the intensity of traffic forwarded in the controlled domain. The functionality aspect of it might be easily augmented and expanded by connecting it to a suite of advanced applications specified for these tasks. The controller itself works as a proxy of sort, between the rules designed by these applications and the network they work on. Of course, an "out-of-the-box" controller has already a number of preinstalled applications working with it, which provide necessary tools to manage and operate the network. SDN architectures allow for the controller to work as a single entity or to share its objective in a decentralized cluster consisting of a number of controllers, which either perform the same functionality, sharing the workload or are tasked with operating within different areas of the network. Authors of this paper utilize the single controller approach.

In most cases, the SDN controller is capable of working in both proactive and reactive mode. The first mode is akin to the legacy networks, in which the administrators of the network designed the rules for management, later to install them by configuring its equipment, e.g., load balancers, switches and routers.

The reactive mode works in a way that allows the controller to react to the changes in the network. It is achieved by making the controller listen for events that denote a necessity of changing the flow rules of its domain. In case of OpenFlow it is done simply by handling packet-in messages which are send to the controller whenever a switch receives a packet for a flow, which it does not know how to handle. All the rules are stored in the switch in "Flow Tables". By default, if a switch receives a packet that does not match any entry in its flow tables, it is encapsulated into a packet-in message and rerouted to the controller. The controller, thanks to its application, checks the packet with its store, where it keeps designed rules of forwarding. If an appropriate rule exists, the packet is returned to the network and the rule is installed into the switch's flow table. This of course means, that the whole operation has to take as little time as possible to not impose any significant delay in the realization of the service.

The controller in the described testbed was limited to reactive mode. The rules in the network were devised by the default mechanisms of the controller. In this case based on the implementation of spanning tree protocol (STP) [37] and controller's knowledge of the topology – which is updated thanks to periodical link layer discovery protocol (LLDP) [37] message handling.

The Floodlight controller [37] was chosen for this solution. Due to the fact, that it allowed to easily tweak with its specification, which was necessary for the experimental environment. It was decided to work in reactive mode, as it was necessary to change how Floodlight installed the rules. To achieve that, authors have reconfigured the module responsible for reactive mode. In the next subsection author present the details on the configuration of the domain and how the measurement was done in it.

It is important to highlight that the controllers do differ from each other and how they are designed may have an impact on the obtained experimental results. This paper does however focus on the "black box approach" trying to distinguish limitations of any controllers, no matter the inner logic that is behind their workings. Papers that provide further insight – by comparing controllers – into this subject are already available to the research community [22], [28].

What is more, the delay imposed by the controller might be tweaked around by focusing on optimizing the code of the software that the controller de facto is. In case of the controller, it is possible to distinguish at least three modules that might impose the delay, them being: Receive Queue, Processing Queue and Sent Queue. This paper focuses on the controller as a black box and do not delve into the details

of its implementation, but research into the matter open an interesting field for further research.

4) MEASUREMENT SETUP

Firstly, the controller had to be reconfigured to prepare it for the measurements. In it, the default matching rules were changed, to make sure that each flow was identified not only by the source and destination IP addresses, but also by transport protocol (either UDP or TCP) and the source and destination ports. The second change was done to the default timeout. For a reactive flow rule, it was changed to last 1 second. (This is the minimal value for timeout allowed in OpenFlow configuration.) Thanks to that, the configuration created a scenario in which a flow responsible for handling a VoIP connection can be removed from the switch during the connections OFF period.

To generate workload for the controller, a single switch was added to the domain. Controller workload generator was connected to this node, which generated a constant stream of packets, providing additional workload for the controller. This single switch through which the packet is send has a high priority flow rule pre-installed, which action is set to sending incoming packets to the controller. It is done by encapsulating the packet in an OpenFlow packet-in message, which is then send to the controller. The controller has to handle such a message and act accordingly. In this paper's case, as this rule should never retire from the switch, the controller only lookups for a matching rule, then stops working on the packet and sends it back to the network. This means, that some of the controller's resources have to be utilized for some period of time for this task. The more packets the controller workload generators generate, the more resources and time controller has to delegate. By changing the number of packets generated by the controller workload generator it was possible to regulate controller's workload as an experiment parameter.

The generator software has been installed on both the generator's, as well as emulator's PCs. For the traffic generation, authors have prepared a number of C++ scripts, which run the MGEN software with different parameters, allowing for simultaneous launch of many MGEN processes, each tasked with generation of a different flow.

The script tasked with VoIP traffic creates n_{VoIP} flows. This has been achieved by parallel running of n_{VoIP} different MGEN processes. Before launching the flow, the script would randomly pick a number from range <0.001; 2.000> (with a 0.000001 resolution), which corresponded to values from 1 ms, to 2 s for all of the VoIP flows. This number was the time between the script launch and the beginning of a flow generation. Each of the flows have been designed as ON/OFF UDP traffic (in MGEN it is achieved by selecting the BURST pattern of packet generation) with a 1.5 s average for OFF period and 1 s average for ON period (both distributions were exponential). Each flow had a different UDP source and destination ports and thanks to that (and the mentioned earlier tweak in the controller's behavior) a separate flow rule has been installed in each of the switches for each of these flows.

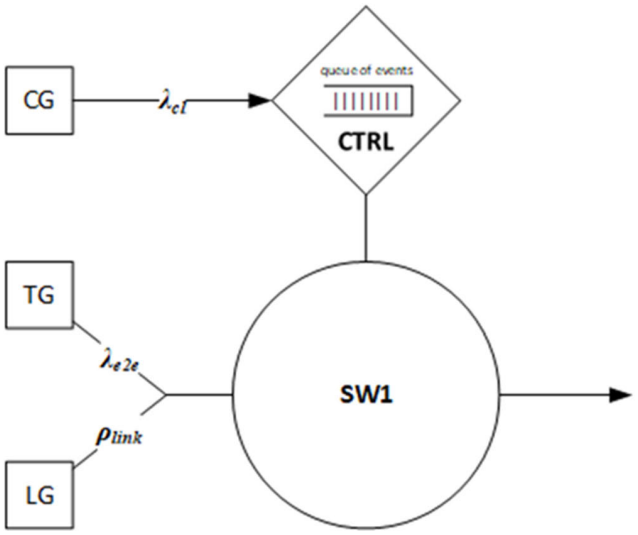

FIGURE 6. Objectives of the generators.

A single packet of this flow corresponds in size to a 20 ms long G.711 voice sample encapsulated into UDP/IP packet.

The script tasked with video traffic created n_{VOD} flows. This has been achieved by parallel running of n_{VOD} different MGEN processes. It picks a start time the same way as in the case of VoIP traffic. The MGEN has been set to replicate the given PCAP file input into appropriate UDP packets with the same size and time relation between them, as in the case of the original captured packets. Just like in the case of the VoIP rules, here as well each of the flows have different source and destination UDP ports.

The script tasked with data traffic created n_{FTP} flows. This has been achieved by parallel running of n_{FTP} different MGEN processes. It picks a start time the same way as in the case of VoIP traffic. The MGEN has been set to replicate the given PCAP file input into appropriate TCP packets with the same size and time relation between them, as in the case of the original captured packets. The TCP source and destination flows were different for each of the flows, just like in the case of VoIP and video flows.

The measurement procedure lasted 30 minutes per iteration and required launching the aforementioned generators:

1) Controller Workload Generator – to provide additional workload for the controller.
2) Three Link Workload Generators – to provide additional workload for the in-between switch links.
3) Scripts for three traffic types on Traffic Generator – to provide the actual teletraffic utilized for the benchmarking procedure.

Fig. 6 gives additional insight into the placement and objectives of the generators.

CG generates a current of events that are enqueued in the controller. This increases its workload, giving a parameter to steer during the measurement process. TG generates actual teletraffic traversing the domain. Traffic from TG, as well as streams from LGs share the same link. In case of the LGs,

their streams are generated to offer additional workload for the links themselves to emulate real-life network processes.

The exact description of the chosen parameters and the obtained results are provided in Section IV of this paper.

IV. MEASUREMENTS AND ANALYSIS

In the framework described in Section III a number of measurements have been conducted to validate influence of the OpenFlow-driven SDN architecture on the QoS of different teletraffic.

Before the measurement procedure started, the authors had to make sure that the framework was working as intended. To verify that, four tests were conducted, which objective was to check if the scenarios of message handling in the proposed testbed complied with the assumptions for OpenFlow. Firstly, during the capture of the packets it has been checked if the time that the controller took to handle a packet-in message was included in the end-to-end delay of that packet. Secondly, it has been verified, if all the necessary steps for OpenFlow packet handling have been undertaken, by checking if such events consisted of all the required protocol messages, this being Packet-In, Flow Mod and Packet-Out. Thirdly, the existence of the required flow rules in the framework's switches has been observed and if their match and actions rules were installed properly. Lastly, authors have validated these flow rules' *n_packets* counters (which counts the number of packets that were matched against the rule) and *time* (which informs about how long the rules have been installed in the table) fields, by checking if their values changed with time accordingly to the modeled scenarios. All four of these checks were successfully passed by the model.

For the measurement procedure the following considerations have been made:

1) The links in-between the nodes had 1 Gb/s bandwidth.
2) Traffic Generator (TG) provided traffic load equal to around 10% of the link bandwidth (a total bitrate of approximately 100 Mb/s). The parameters of this traffic were: $n_{flows} = 264$, $n_{VoIP} = 250$ (a bit rate equal to 16 Mb/s), $n_{VOD} = 10$ (with summed bit rate of 39.79 Mb/s), $n_{FTP} = 4$ (the total summed bitrate was 41.738 Mb/s). The details on the streams included in this traffic are provided in Tab. 2.
3) A single measurement lasted 30 minutes.
4) In a single measurement two parameters are set: ρ_{link} and λ_{cl}.

The measurement procedure has been conducted 90 times. Which included 5 measurement intervals per set of variables ρ_{link} and λ_{cl}. These set values are presented in Tab. 3.

Authors focus on validating the IPTD as the essential QoS metric. Papers on the subject of QoS in SDN mostly benchmark IPTD and IPLR. In case of the author's research there were no packet losses observed in the testbed.

The measurements take under consideration how the link loads and the controller's workload influence the time parameters of the traffic flows traversing through the domain.

TABLE 2. Flows utilized during the measurements.

Flow	Bandwidth	Additional info
250 VoIP flows	≈ 64 kb/s per flow	
VOD flow 1	3.349 Mb/s	Hurst: 0.7565
VOD flow 2	1.985 Mb/s	Hurst: 0.7965
VOD flow 3	3.761 Mb/s	Hurst: 0.7391
VOD flow 4	4.843 Mb/s	Hurst: 0.7865
VOD flow 5	3.565 Mb/s	Hurst: 0.8140
VOD flow 6	5.656 Mb/s	Hurst: 0.6957
VOD flow 7	4.43 Mb/s	Hurst: 0.7652
VOD flow 8	4.712 Mb/s	Hurst: 0.6996
VOD flow 9	4.185 Mb/s	Hurst: 0.7739
VOD flow 10	3.331 Mb/s	Hurst: 0.7721
FTP flow 1	10.633 Mb/s	Hurst: 0.3662
FTP flow 2	10.618 Mb/s	Hurst: 0.3652
FTP flow 3	10.491 Mb/s	Hurst: 0.3014
FTP flow 4	9.936 Mb/s	Hurst: 0.3524

TABLE 3. Types of traffic utilized in the paper.

Set	ρ_{link}	λ_{cl}
1	0	0
2	0	250
3	0	500
4	0	750
5	0	1000
6	0	1250
7	0.35	0
8	0.35	250
9	0.35	500
10	0.35	750
11	0.35	1000
12	0.35	1250
13	0.7	0
14	0.7	250
15	0.7	500
16	0.7	750
17	0.7	1000
18	0.7	1250

It has been noticed that for data and video streams there was only a single packet handling event per stream. This came to no surprise, as the time intervals between consecutive packets was lower than 1 second. This means that the flow rules for these streams were never purged from the flow tables. With that in mind the authors have decided to minimize their importance in the validation of the model, as their end-to-end delays would be only influenced by the link loads and not the controller's workload.

However, the case for voice traffic flows was significantly different. The generated traffic was emulating artificial speech and included times of "silence" that came from emulating time in between talk sprouts, during which the speaker does not talk. This led to an enormous amount of time periods lasting longer than 1 second that created opportunities for the flows to be handled by the controller. The set of results for voice traffic was chosen as a manner of representing the influence of controller's performance on the SDN architecture.

From these results authors selected parameters for validation of the performance. These were:

1) mean end-to-end delay of packets traversing the domain $E(t_{e2e})$,
2) mean end-to-end delay of packets traversing the domain, without packets send to the controller $E(t_{e2e\_without\_cl})$,
3) mean time of the controller handling a packet $E(t_{ch})$.

To present and analyze the results, authors have decided to present them in a manner of forms:

1) line chart presenting the function of $E(t_{e2e}) = f(\lambda_{cl})$, for different ρ_{link},
2) distribution of $E(t_{e2e})$ for different λ_{cl} and ρ_{link},
3) distribution of $E(t_{e2e})$ of packets sent to the controller for different λ_{cl} and ρ_{link},
4) percentage of packets with t_{e2e} higher than 1 ms for different λ_{cl} and ρ_{link},
5) percentage of packets with t_{e2e} higher than 100 ms for different λ_{cl} and ρ_{link},
6) percentage of packets sent to the controller with t_{e2e} higher than 1 ms for different λ_{cl} and ρ_{link},
7) percentage of packets sent to the controller with t_{e2e} higher than 100 ms for different λ_{cl} and ρ_{link},
8) ratio of packets sent to the controller to total number of packets with t_{e2e} higher than 1 ms for different λ_{cl} and ρ_{link},
9) ratio of packets sent to the controller to total number of packets with t_{e2e} higher than 100 ms for different λ_{cl} and ρ_{link}.

The above has been conducted with the link length settings in the emulator set to 0, which means that the time of propagation through the links in the domain are not taken into consideration.

Firstly, the results were presented on the graph showing the relation between the delay and the controller's workload. This has been drawn for three different link loads in Fig. 7.

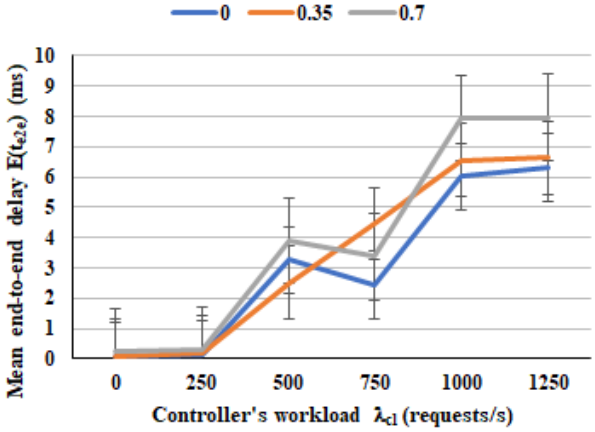

FIGURE 7. Mean end-to-end delay.

Fig. 7 shows the dependency between increasing controller's workload and the increase in the mean end-to-end delay. The differences between the values for different workloads are not that visible for low workloads of 0 or 250 requests per second, meaning that the controller works without any issues, no matter the link loads at these points. The differences start to be visible with the increase of workload to 500 and 750 requests per second. Mean end-to-end delay increases to values from 2 to 4 ms, the biggest increase is visible at the 1000 and 1250 requests per second point, where it reaches values from 6 to 8 ms. This outcome is the result of the increase in packets handling time in the controller, as presented in (6). With the increase of the number of packets to be handled (regulated with λ_{cl}) the mean time of being handled by the controller increases as well. Additionally, with the increase of link workload increases number of packets that land in switches' buffers and with the queue increasing in size, time spend in the switch increases as well, which leads to the increase in total end-to-end delay.

The authors decided to validate if there is an indication of a phenomena which might lead to the sudden increase in delay of the packets traversing the SDN domain.

It is crucial to distinguish two different packets groups: packets which were handled by the controller and packets that were not. A packet handling by the controller is a part of the end-to-end delay, as indicated in formula (5). Authors decided to verify how much of an impact on these two packets groups have λ_{cl} and ρ_{load}. To delve deeper into understanding of the measured values, the authors decided to present how big part of the end-to-end delay is the effect of being handled by the controller. Two graphs have been charted. One of them (Fig. 8) presents the end-to-end delay of the packets that were not sent to the controller, the second (Fig. 9) presents the time required for the controller to handle a packet send to it.

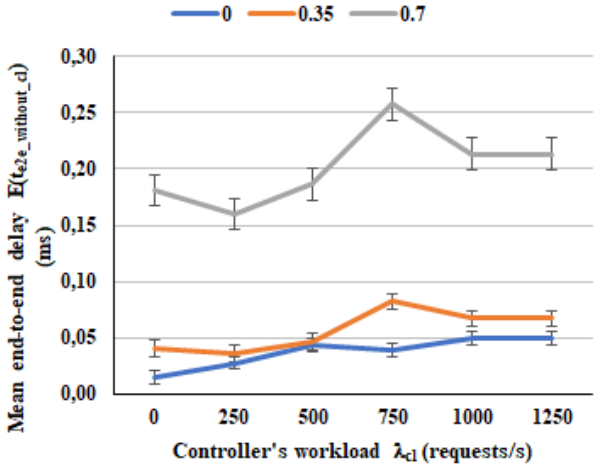

FIGURE 8. Mean end-to-end delay of packets not sent to the controller.

As seen in Fig. 8, the mean end-to-end delay of packets that were not send to the controller are of small values. The link load seems to impact the delays as there is a noticeable increase in the values between 0.7 link load and the other two, however all of the values themselves are diminutive, not even reaching half of a millisecond.

These results were expected. As noted in the previous paragraphs, with the increase of ρ_{load}, number of packets awaiting forwarding by the nodes increases, as in (3). The difference in delay between values of 0 and 0.35 is negligibly low. However, the 0.7 link load has a distinguishable increase in end-to-end delay.

When compared with the results charted in Fig. 7, authors notice that delay imposed by the switch plane itself has a small effect on the total end-to-end delay, the impact of controller's workload doesn't change the values that much. Suggesting that the bigger part in influencing the end-to-end delay is played by the controller's handling time.

To prove this point, a graph was charted, presenting the impact of the controller's workload on the time required by the controller to handle a packet-in event. The graph is presented in Fig. 9.

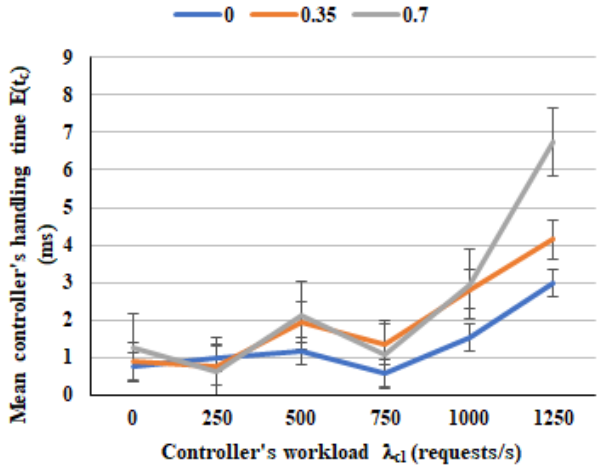

FIGURE 9. Mean controller's handling time.

With the increase of number of requests to handle, the values for mean controller's handling time increase as well. This is not surprising, after all with the increase of workload the time of waiting for being handled must also increase.

What is of note in Fig. 9 is the difference in waiting time between different link loads for the highest workload value. It is most likely, that the time of being handled by the switch are also influencing the results. After all, the packet that requires attention from the controller has to be handled by the switch first. More important are the values themselves when Fig. 7 is taken into context. When comparing a packet mean end-to-end time with the time of a packet being handled by the controller, it can be concluded that the latter takes the majority of time needed for the packet to be sent over, which means that efficiency of the controller's handling time is crucial for the process of sending the packet to its destination.

When analyzing the results, one must take under consideration, that in this environment, the times of propagation in the domain were equal to zero. Authors begin with the presentation of results for such a situation. Later on in this section, a different set of results for situation involving an existing distance between nodes are presented.

To validate the influence of the controller's performance on the variation of delay, the authors decided to check the distribution of end-to-end delay values. The histograms of this distribution are charted in Fig. 10, 11 and 12.

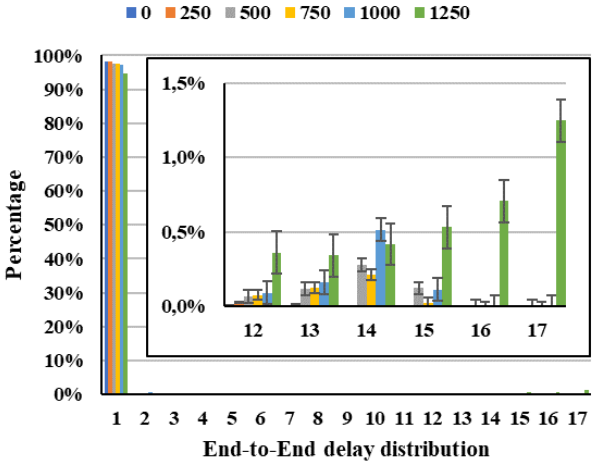

FIGURE 10. End-to-End delay distribution of all packets for $\rho_{link} = 0$ and different λ_{cl}.

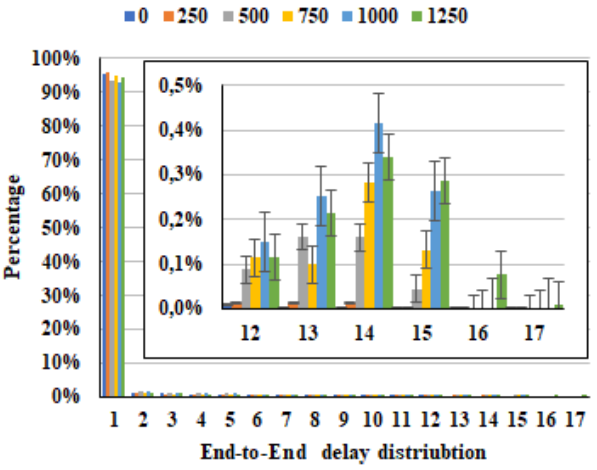

FIGURE 11. End-to-End delay distribution of all packets for $\rho_{link} = 0.35$ and different λ_{cl}.

Before delving into analysis of the histograms, some explanation is required. The numbers assigned to the x-axis are the ranges of end-to-end delays. Tab. 4 presents the values of the ranges.

Most of the packets end-to-end delays have values of lower than 250 μs. With the increase in link load the percent decreases. With link load of 0 almost 100% of the packets have lowest possible delays, no matter the controller's workload, with 0.35 the number decreases to 95% and with 0.7 they reach 90%. With the decrease of that value increases the number of packets with delays of higher values.

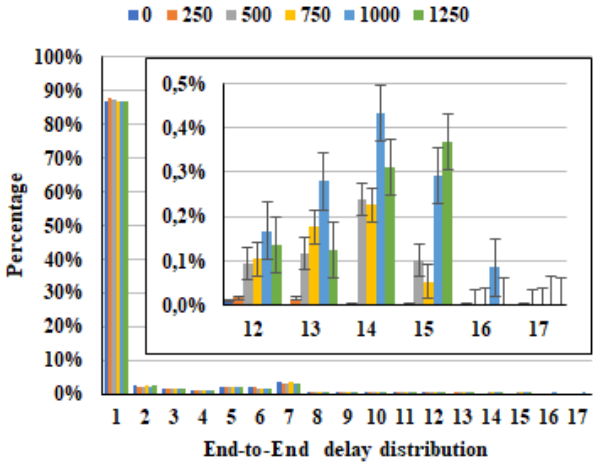

FIGURE 12. End-to-End delay distribution of all packets for $\rho_{link} = 0.7$ and different λ_{cl}.

TABLE 4. Types of traffic utilized in the paper.

Number from x-axis	Delay range
1	(0; 250 µs>
2	(250 µs; 500 µs>
3	(500 µs; 750 µs>
4	(750 µs; 1 ms>
5	(1 ms; 1.5 ms>
6	(1.5 ms; 2 ms>
7	(2 ms; 5 ms>
8	(5 ms; 10 ms>
9	(10 ms; 25 ms>
10	(25 ms; 50 ms>
11	(50 ms; 100 ms>
12	(100 ms; 250 ms>
13	(250 ms; 500 ms>
14	(500 ms; 1 s>
15	(1 s; 2.5 s>
16	(2.5 s; 5 s>
17	> 5 s

For clarity, authors decided to implement a ''zoom-in'' into some values of the packets' delays, to show the percentage of packets whose delays went above the value of 100 ms. It is clearly visible, that the increase of the number of packets with such delays increases when link load increases above 0. In these cases, the number of packets with delays higher than 100 ms reach about 1% of all of the packets for 1250 requests per second of controller's workload.

To validate the results further, the authors decided to present also the distribution for packets send only to the controller. To see how many of them have delays above the acceptable value of 100 ms. This have been charted in Fig. 13, 14 and 15.

Packets that were analyzed belonged to flows representing VoIP sessions with crucial limitations on acceptable delay values. The packets that were sent over to the controller had an increased delay overall. This becomes even more noticeable the bigger the controller's workload. This in conjunction with the observations from charts in Fig. 7-9 paint a grim picture. Even though the end-to-end delay may increase but not break the acceptable values, the percentage of packets that break the threshold is high enough. This might suggest that with the increase in controller's workload the percentage might increase as well.

The more the percentage of packets handled by the controller increases, the bigger the mean value of overall end-to-end delay is going to be.

The physical limitations of the testbed did not allow the authors to validate the measurements with higher workloads for the controller. However, even with the obtained results it is to note, that with the increase in the workload the potential for breaching the acceptable values of IPTD is increasing noticeably. To make sure that this assessment is correct, the authors decided to validate how many packets sent to the controller had delays above 1 and 100 ms and to check how many overall packets with delays above 1 and 100 ms were sent to the controller beforehand.

Firstly, authors calculated the number of packets that were send to the controller. From this group packets with delays larger than 1 ms have been selected. The latter number was divided by the previous one and charted in Fig. 16. Fig 17 presents the similar done for delays above 100 ms.

It seems from Fig. 16 that most of the time the controller imposes at least 1 ms of delay, as significant chunk of packets handled by the controller have a delay higher than 1 ms. The differences between different λ_{cl} do not influence the percentages noticeably, as the percentage of packets with delays higher than 1 ms depending on controller's workload oscillates around 80-90%.

The difference starts to be seen when the authors analyze delays of over 100 ms. From Fig. 17 authors conclude that for low controller workloads the increase of a delay above 100 ms is a rare occurrence. With its increase, the increase in the number of packets with undesirable delay is highly noticeable. A jump from 250 to 500 requests per second leads to the increase in 20%, from 500 to 1000 to another 20%.

The above proves, that packets with end-to-end delays out of the question are mostly the effect of staying too long in the controller. This is also a suggestion of what is the real impact of the increase in controller's workload on the end-to-end delays in an SDN domain.

Second checkup was concerned with how many packets that break the 1 ms and 100 ms limits for end-to-end delays were handled beforehand by the controller. Authors counted all packets that had delays above 1 ms. Then from that group, authors counted all packets that were send to the controller. The latter value has been divided by the former (Fig. 18). The same procedure has been conducted for packets with delays above 100 ms (Fig. 19).

In Fig. 18 shows that for delays higher than 1 ms, the increase in link workload imposes an adverse effect. In case of the highest link workload, only a fifth of all packets with delay above 1 ms were the ones handled by the controller. This is even more prominent with results charted in Fig. 19. A significant majority of packets with delay above 100 ms were the ones handled by the controller.

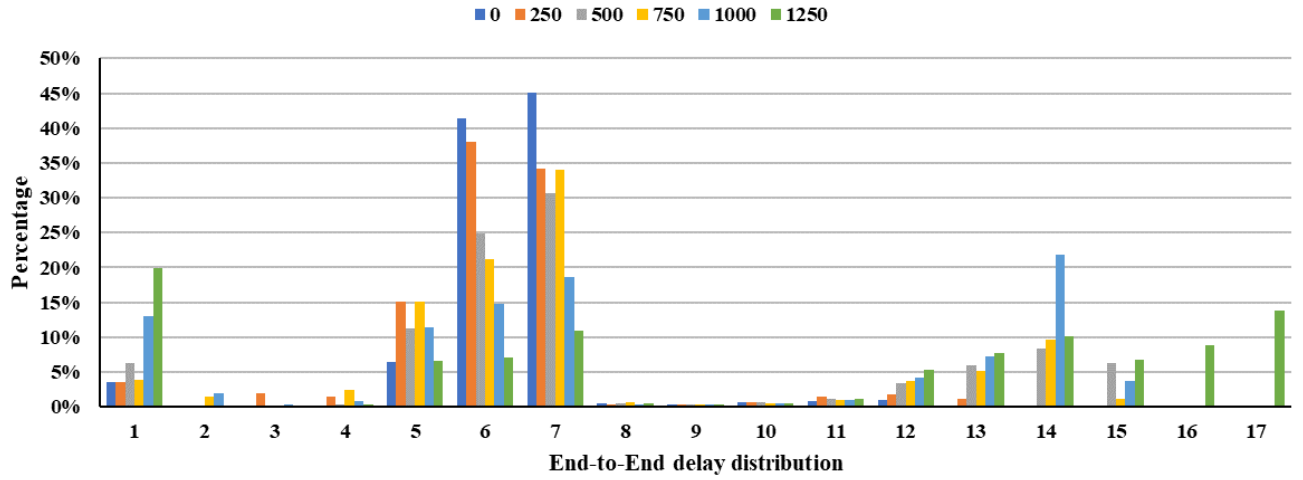

FIGURE 13. End-to-End delay distribution of packets sent to the controller for $\rho_{link} = 0$ and different λ_{cl}.

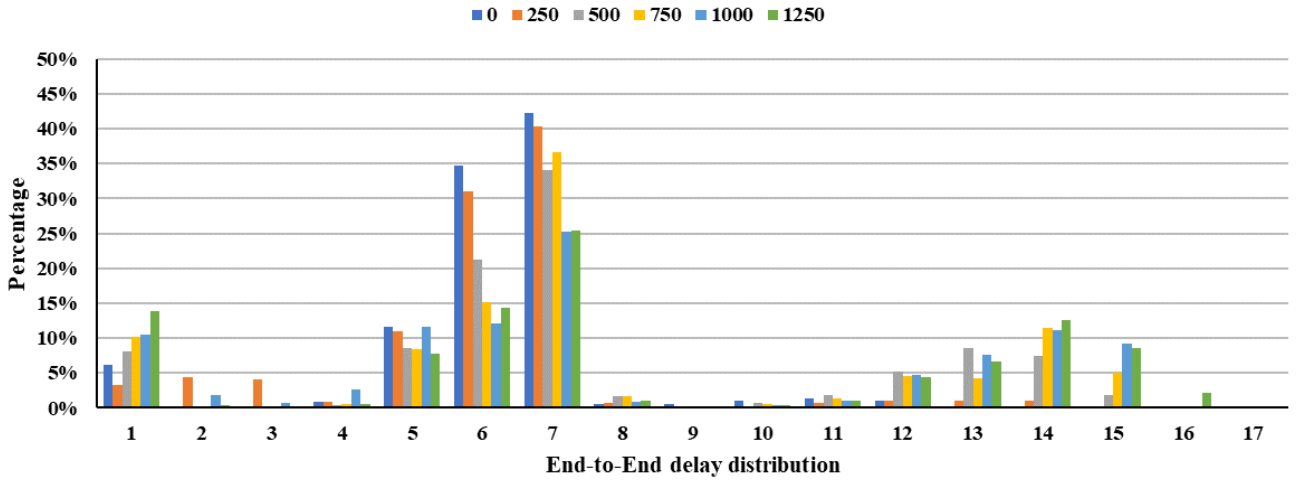

FIGURE 14. End-to-End delay distribution of packets sent to the controller for $\rho_{link} = 0.35$ and different λ_{cl}.

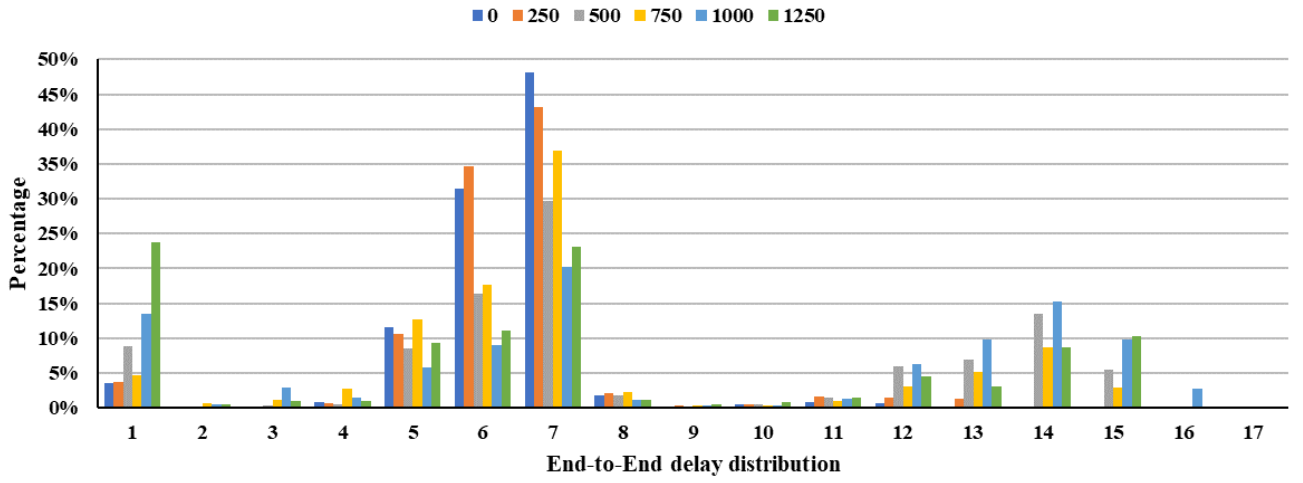

FIGURE 15. End-to-End delay distribution of packets sent to the controller for $\rho_{link} = 0.7$ and different λ_{cl}.

These suggest that controller does in fact influence the delays of packets from certain flows in a manner that is highly unwelcome.

A model that does not include the distances between network entities is a far cry from the actual telecommunications network. After all, distances between network elements and

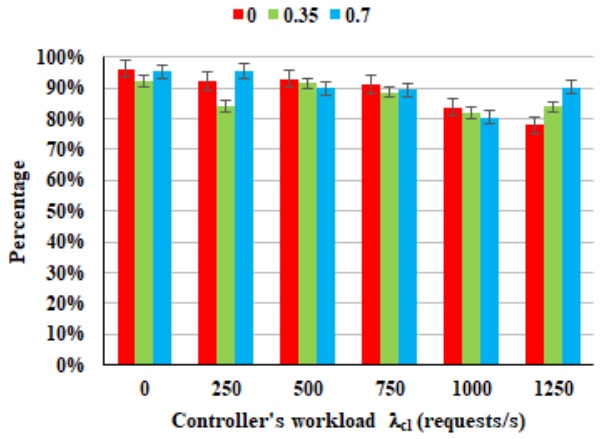

FIGURE 16. Percentage of packets sent to the controller with t_{e2e} > 1 ms for different ρ_{link} and λ_{cl}.

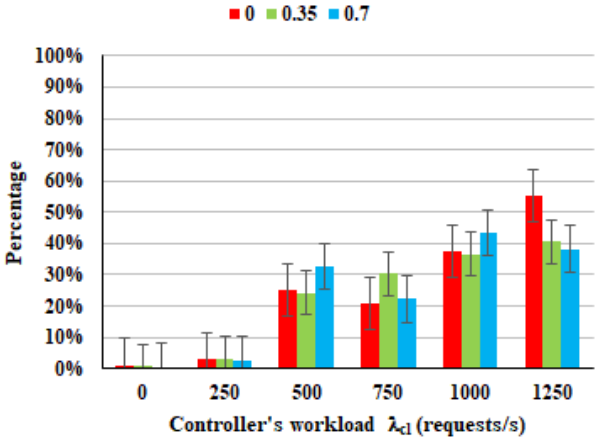

FIGURE 17. Percentage of packets sent to the controller with t_{e2e} > 100 ms for different ρ_{link} and λ_{cl}.

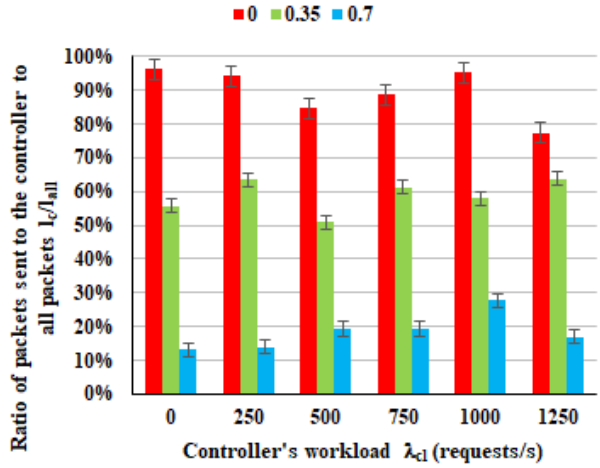

FIGURE 18. Ratio of packets sent to the controller to all packets with t_{e2e} > 1 ms for different ρ_{link} and λ_{cl}.

their location in relation to each other may have an impact on the time of propagation by the transmission systems in the network.

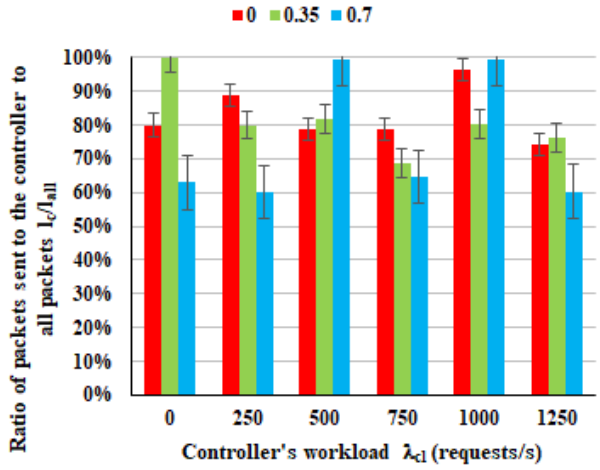

FIGURE 19. Ratio of packets sent to the controller to all packets with t_{e2e} > 100 ms for different ρ_{link} and λ_{cl}.

Taking that into account, as seen in (4), it can be concluded that with too far of the distance between the entities in this network, the delay may increase. Especially if the controller is too far, as the time of propagation from the switch to the controller and vice versa are part of the equation (6) and (7). This has already been analyzed in [38].

Authors have already committed to theoretical analysis of that phenomena in previous sections of this paper (4). With that in mind, the obtained results presented in previous paragraphs have been evaluated again with inclusion of the distances between SDN switches and the controller.

Network from Fig. 2, which the authors utilized in the emulation process, have been adjusted by including the distances between the SWn switches. Each inter-switch link has been given a length of 100 km. Additionally, the links between the SWn and CTRL controller has also been determined to have a length of 100 km each. Each link is set to be an optical fiber connection, which imposes a delay of 5 μs per kilometer. With a length of 100 km for each link, every connection between elements of this network will now add a total of 500 μs of propagation delay.

Adjusting the results to include the propagation delay has been conducted thoroughly and with caution. First of all, every single packet's delay has been increased by 1.5 ms. That is because, each packet had to travel through three inter-switch links. Each such 100 km long link added 500 μs of delay. Secondly, some of the packets have their delay increased by additional 1 ms. These packets were selected based on, whether they have been transferred to the controller when there were no matching rules for them in the entry switch SW1. This value came from the links between the SW1 and the CTRL. A packet has to be sent to the controller for handling and then returned to the switch it was sent from, as seen in Fig. 3. As these links were also 100 km in length, going through these connections once add 500 μs of delay. In the described case each handled by the CTRL packet traverses this connection twice.

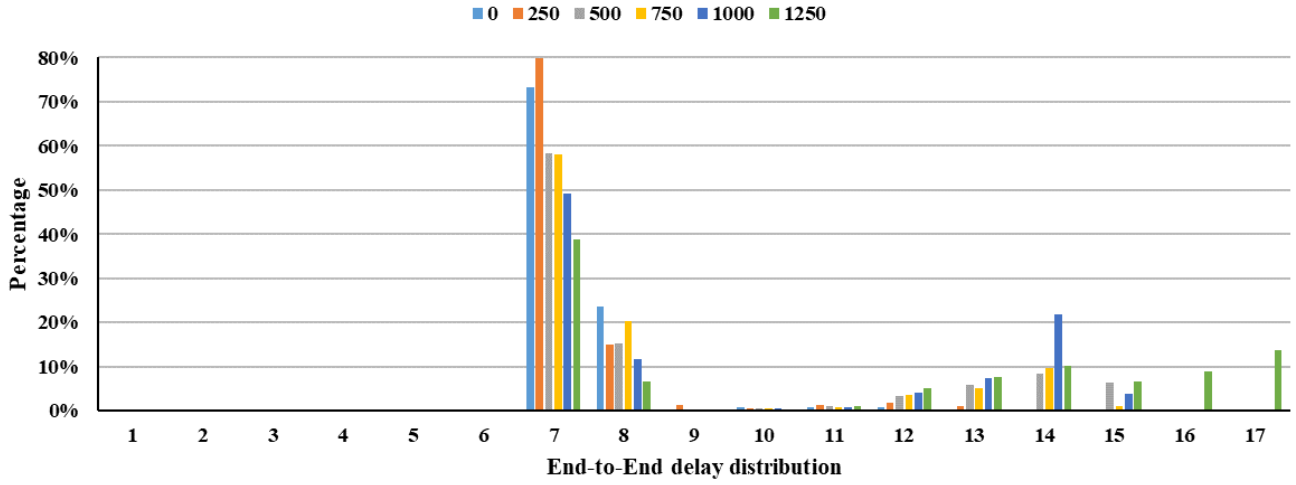

FIGURE 20. End-to-End delay distribution of packets sent to the controller for $\rho_{link} = 0$ and different λ_{cl}, including propagation.

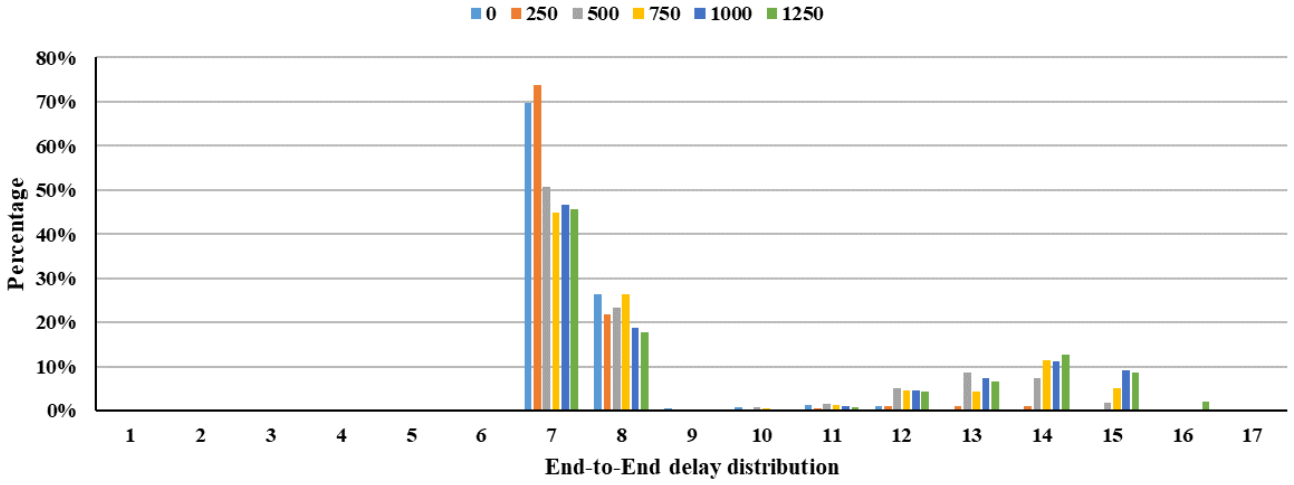

FIGURE 21. End-to-End delay distribution of packets sent to the controller for $\rho_{link} = 0.35$ and different λ_{cl}, including propagation.

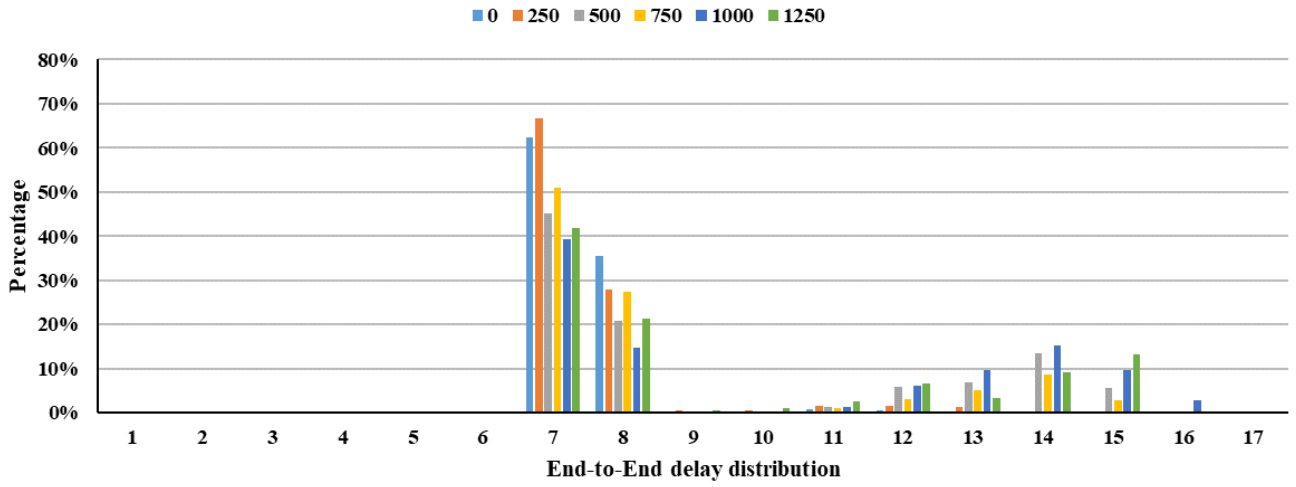

FIGURE 22. End-to-End delay distribution of packets sent to the controller for $\rho_{link} = 0.7$ and different λ_{cl}, including propagation.

Including the presented modifications to the results, the authors provide how does the performance of the controller influence Quality of Service with the time of propagation in mind.

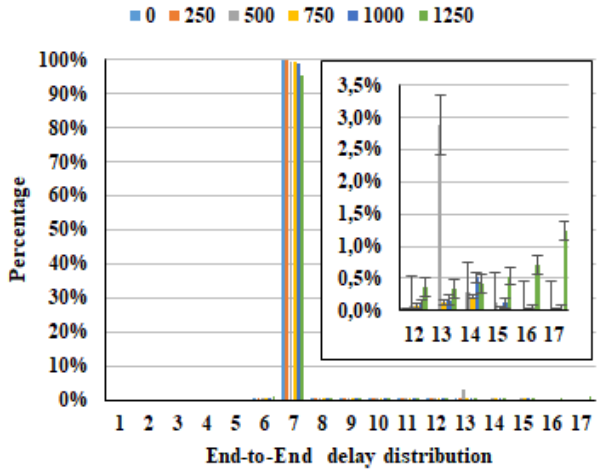

FIGURE 23. End-to-End delay distribution of all packets for $\rho_{link} = 0$ and different λ_{cl}, including propagation.

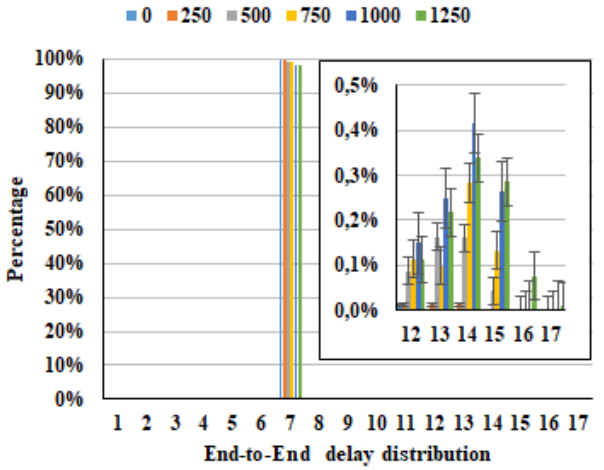

FIGURE 24. End-to-End delay distribution of all packets for $\rho_{link} = 0.35$ and different λ_{cl}, including propagation.

Discussing them require comparison with previously charted histograms, that did not include the propagation delays.

Firstly, the authors provide histograms for the end-to-end delays of packets that have been sent to the controller for handling. These are provided in Fig. 20, 21 and 22.

Fig. 20, 21 and 22 provide the same data as Fig. 13, 14 and 15, but with the inclusion of the effect of distance on the delays. Before the inclusion, the charts had a 3-modal shape with most of the values concentrating in the range from 2 to 10 ms. After the addition of propagation time the shape of the distribution changed to 2-modal.

In both situations we notice that around 10% of the packets sent to the controller have an end-to-end delay of above 100 ms, which is considered to be the threshold of acceptance when it comes to VoIP connections [36], [37], [38], [39], [40], [41].

To complete the analysis, authors provide the same charts but for all packets, no matter whether they were sent to the controller, or not. They are presented in Fig. 23, 24 and 25.

As in the case of distribution of packets sent to the controller, here as well the shape of the distribution changed, this time by moving the values from ranges 0 to 250 μs into 2 to 5 ms. This combined with the previously existing values in this ranges lead to a concentration of values on a level of over 90%. However, in case of the values increasing beyond 100 ms, the total number of such packets is miniscule, no matter if the propagation is, or is not included in the results.

Above suggests that for the given range of controller workloads the overall quality of VoIP connection should not be impacted in the long run, still there is a number of packets whose delays exceed the acceptable values, when they have to be handled by the controller.

In the next section authors provide their conclusion on the matter experimented on in this section.

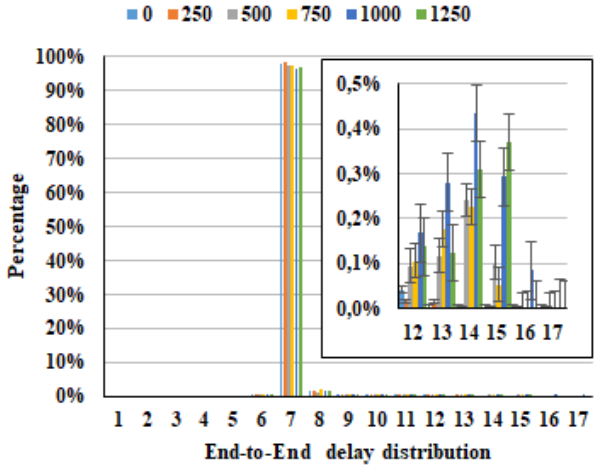

FIGURE 25. End-to-End delay distribution of all packets for $\rho_{link} = 0.7$ and different λ_{cl}, including propagation.

V. CONCLUSION

The designed environment fulfilled the required criteria when it came to conducting the experiments. With the use of hybrid emulated/physical approach a number of results have been obtained. As mentioned in Section III, the testbed stores results in CSV and PCAP file formats, allowing for their further analysis on other research platforms.

The results provided in charts from the previous section give an insight on how the controller performance impacts the flow in context of keeping up with the Quality of Service.

Results were provided only for VoIP connections, as in the case of data and VOD flows, only a single instance of requiring handling by the controller has occurred. The authors conclude that for the modern SDN network based on OpenFlow protocol, the given time-resolution for the times of storing a flow in the switch are enough to not influence QoS for flows that are long living and have short, lower than 1 s intervals in-between the packets. The controller

workload impacts such a flow only at the beginning of the flow emergence in the network.

Results for VoIP streams are more interesting. In case of telephone connections there might be instances in which an intermission between two packets occurring might be beyond 1 s. In such cases the flow has to be reinstalled. Of course, this might be mitigated by prolonging the timer for keeping the flow inside of the flow table. This allows for the flow to exist in the flow table for a longer time than necessary. However, in a highly utilized network with a lot possible sources, destinations and service types, a number of such flows might increase drastically. Network's operator has to include that in a modern network there is a high volatility when it comes to streams ending, emerging and reinstating. With that a problem of storing the instructions for flow handling in a somehow limited space of the flow tables emerges [42].

The next paragraphs will focus on the phenomena observed and noted by the authors.

First of all, the increase of the controller's workload always leads to the increase of average end-to-end delay. The higher the workload is, the higher the expected value of the delay, as seen in Fig. 7, as expected. This issue with SDN architecture is still a crucial point of concern [15].

Secondly, the distance between the controller and the switches has a significant effect on the delay imposed by the controller, as seen in Fig. 20-25. Not only does it increase the mean end-to-end delay value, but it also changes the shape of the distribution of end-to-end delays for all the packets forwarded in the network.

Thirdly, from the distribution charts it is clearly visible that there exists a threshold for both controller's workload and the distance between the switch and the controller, as seen in Fig. 20-25 and with accordance to (6) and (7). Crossing the thresholds lead to mean values exceeding the limits imposed by conforming to QoS requirements.

Fourth, no matter the type of the flow, the workload of the controller or the link loads, the first packet of the flow will always be delayed. The value of the delay depends on the controller's workload and increases with it, as seen in Fig. 9.

Finally, for higher controller's workload an undesired phenomenon occurs in which a second packet of the flow enters the controller for handling while the previous packet is still being handled. With too much of a controller's workload the effect increases in scope, leading to avalanche effect in which the delay from handling of a packet increases the delay of the following packets.

The overall end-to-end delay does seem to be impacted by the controller's workload. This is even more of a complex issue, as the flow might expire even though the telephone connection did not end. The pauses of more than 1 second might occur during a single connection due to the nature of a human conversation, as defined in the ITU-T recommendation [36]. After each of such pauses a necessity to re-establish the flow rule emerge, which adds to the end-to-end delay by repeating the process of being handled by the controller. These events may lead to an unpleasant experience of unstable phone connection. It is also imported to highlight that the impact on the end-to-end delay is minimal in the observed range of the values, with the visible increase of it dependent on the increase of controller's workload it is assured that there is a threshold beyond which the controller's workload starts to impose a noticeable delay.

Listed problems might be mitigated by implementing appropriate resource control mechanism into the SDN architecture. A complete solution would require the controller to recognize flows of high priority that require expediate handling. In other words, a solution would be to implement a class-based handling in the SDN controller, akin to network traffic classes commonly associated with Next Generation Networks. Additionally, a number of rules in the controller should work as a guideline when installing the rules in the network. These should include recognizing the type of traffic belonging to a flow or sources and recipients of traffic that pose higher requirements on the delay values.

With all of the above summarized a field for following research can be established, that includes following experimentation.

First of all, as the authors noted, the observed behavior of the controller's workload might impact an altogether different type of flows, which are the short-living time-constraint flows. A number of IoT devices produce such traffic flows, where they require a fast handling, being sent to the recipient without much of a delay, but also sending a rather small amount of information in just a few packets [43].

Secondly, a wider range for the tested parameters should be incorporated, this is not trivial, as there are constrains from the hardware at the disposal. However, it would be possible to implement the testbed in a simulator. This would allow increasing the range for the variables influencing the results, as well as to mitigate issues coming from constrains derived from physical environment.

Thirdly, an attempt at an implementation of the resource control mechanisms might be due to be conducted to test whether would they influence positively the mean end-to-end delays for flows that have rigid requirements on QoS. Resource control mechanisms may be implemented in either the controller or the switches. Solutions may include, to name just a few: controller monitoring the packet-in events to distinguish the priority of the flows sent to be handled by the controller; switches keeping a backup of a flow rule for flows that require expediate handling and have a tendency to re-emerge further down the timeline.

Authors are aware of the limitations imposed by the testbed. Even though it is emulating an environment as close to a real-life telecommunications network as possible, additional adjustments can be provided. Most of them are the effects of the limiting nature of the hardware utilized in the framework. More powerful computer devices would allow to expand on the range of conditions validated in

the experiments (like controller's workload), or the bandwidth of links used in interconnections between the switches. Authors note that the research conducted in this limited workspace still provides a valuable insight into the nature of SDN controller's performance when faced with dealing with real-life teletraffic use cases. Most papers concerned with controller's performance simplify their work by utilizing requests pipeline that do not emulate the real-life occurrences. When it comes to providing a valid solution for a telecommunications network, engineers should opt for solutions that have been tested for environments that handle teletraffic. In a manner of fact, results presented in this paper provide exactly such insight. What is more the environment is easily portable, so it can be moved to a different hardware suite. Upgrading the hardware opens the doors for advancing the research.

The conclusion of the authors is that the controller does in fact influence the QoS parameters in the tested range of parameters. However, the noticeable influence is more nuanced that simple delay values of the packets. In case of VoIP flows it was indicated that the first packet of the flow is delayed in a significant matter, which increases with the increase of workload of the controller. In case of data or VOD flows this might not lead to a significant decrease in the quality of the service, but it might influence negatively a specific type of streams. Them being streams that are short lived, yet requiring an expedite forwarding through the network. A use case that fits this description is traffic from IoT, the reasoning behind this argumentation being provided in the previous section of this paper.

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[2] R. Horvath, D. Nedbal, and M. Stieninger, "A literature review on challenges and effects of software defined networking," *Proc. Comput. Sci.*, vol. 64, pp. 552–561, Oct. 2015, doi: 10.1016/j.procs.2015.08.563.

[3] *Internet Protocol Aspects—Quality of Service and Network Performance*, document ITU-T Rec. Y.1541, 2006.

[4] R. Narisetty, L. Dane, A. Malishevskiy, D. Gurkan, S. Bailey, S. Narayan, and S. Mysore, "OpenFlow configuration protocol: Lmplementation for the of management plane," in *Proc. 2nd GENI Res. Educ. Exp. Workshop*, Salt Lake City, UT, USA, Mar. 2013, pp. 66–67, doi: 10.1109/GREE.2013.21.

[5] M. Beshley, M. Seliuchenko, O. Panchenko, O. Zyuzko, and I. Kahalo, "Experimental performance analysis of software-defined network switch and controller," in *Proc. 14th Int. Conf. Adv. Trends Radioelecrtronics, Telecommun. Comput. Eng. (TCSET)*, Lviv, Ukraine, Feb. 2018, pp. 282–286, doi: 10.1109/TCSET.2018.8336203.

[6] S. Asadollahi and B. Goswami, "Experimenting with scalability of floodlight controller in software defined networks," in *Proc. Int. Conf. Electr., Electron., Commun., Comput., Optim. Techn. (ICEECCOT)*, Mysuru, India, Dec. 2017, pp. 288–292, doi: 10.1109/ICEECCOT.2017.8284684.

[7] L. Mamushiane, A. Lysko, and S. Dlamini, "A comparative evaluation of the performance of popular SDN controllers," in *Proc. Wireless Days (WD)*, Dubai, UAE, Apr. 2018, pp. 54–59, doi: 10.1109/WD.2018.8361694.

[8] I. Z. Bholebawa, R. K. Jha, and U. D. Dalal, "Performance analysis of proposed network architecture: OpenFlow vs. traditional network," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, pp. 30–39, Mar. 2016, doi: 10.6084/m9.figshare.3153850.

[9] F. Laassiri, M. Moughit, and N. Idboufker, "Evaluation of the QoS parameters in different SDN architecture using Omnet 4.6++," in *Proc. 18th Int. Conf. Sci. Techn. Autom. Control Comput. Eng. (STA)*, Monastir, Tunisia, Dec. 2017, pp. 690–695, doi: 10.1109/STA.2017.8314976.

[10] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in SDN," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 472–503, 1st Quart., 2020, doi: 10.1109/COMST.2019.2935453.

[11] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, and V. Vercellone, "Evaluating the SDN control traffic in large ISP networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 5248–5253, doi: 10.1109/ICC.2015.7249157.

[12] B.-y. Yu, G. Yang, and C. Yoo, "Comprehensive prediction models of control traffic for SDN controllers," in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, Montreal, QC, Canada, Jun. 2018, pp. 262–266, doi: 10.1109/NETSOFT.2018.8460111.

[13] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, Aug. 2011, pp. 254–265, doi: 10.1145/2018436.2018466.

[14] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, Aug. 2010, doi: 10.1145/1851275.1851224.

[15] B. Isyaku, M. S. M. Zahid, M. B. Kamat, K. A. Bakar, and F. A. Ghaleb, "Software defined networking flow table management of OpenFlow switches performance and security challenges: A survey," *Future Internet*, vol. 12, no. 9, pp. 147–176, Aug. 2020, doi: 10.3390/fi12090147.

[16] *Traffic Flow Types for Testing Quality of Service Parameters on Model Networks*, document ITU-T Rec. Q.3925, 2012.

[17] C. Metter, M. Seufert, F. Wamser, T. Zinner, and P. Tran-Gia, "Analytical model for SDN signaling traffic and flow table occupancy and its application for various types of traffic," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 3, pp. 603–615, Sep. 2017, doi: 10.1109/TNSM.2017.2714758.

[18] Y. Liu, Y. Li, Y. Wang, Y. Zhang, and J. Yuan, "On the resource trade-off of flow update in software-defined networks," *IEEE Commun. Mag.*, vol. 54, no. 6, pp. 88–93, Jun. 2016, doi: 10.1109/MCOM.2016.7498093.

[19] U.S. Naval Research Laboratory. *Multi-Generator (MGEN) Network Test Tool*. Accessed: Nov. 21, 2023. [Online]. Available: https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/MGEN/

[20] The Tcpdump Group. *TCPDUMP & LIBPCAP*. Accessed: Nov. 21, 2023. [Online]. Available: https://www.tcpdump.org/

[21] Mininet Project Contributors. *Mininet*. Accessed: Nov. 21, 2023. [Online]. Available: http://mininet.org/

[22] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, and S. A. Mehdi, "An architectural evaluation of SDN controllers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Budapest, Hungary, Jun. 2013, pp. 3504–3508, doi: 10.1109/ICC.2013.6655093.

[23] A. Shalimov, D. Zuikov, and D. A. Zimarina, "Advanced study of SDN/OpenFlow controllers," in *Proc. Central Eastern Eur. Softw. Eng. Conf.*, 2013, pp. 1–6.

[24] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible OpenFlow-controller benchmark," in *Proc. Eur. Workshop Softw. Defined Netw.*, Darmstadt, Germany, Oct. 2012, pp. 48–53, doi: 10.1109/EWSDN.2012.15.

[25] M. Jarschel, C. Metter, T. Zinner, S. Gebert, and P. Tran-Gia, "OFCProbe: A platform-independent tool for OpenFlow controller analysis," in *Proc. IEEE 5th Int. Conf. Commun. Electron. (ICCE)*, Danang, Vietnam, Jul. 2014, pp. 182–187, doi: 10.1109/CCE.2014.6916700.

[26] Y. Zhao, L. Iannone, and M. Riguidel, "On the performance of SDN controllers: A reality check," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, San Francisco, CA, USA, Nov. 2015, pp. 79–85, doi: 10.1109/NFV-SDN.2015.7387410.

[27] P. Isaia and L. Guan, "Performance benchmarking of SDN experimental platforms," in *Proc. IEEE NetSoft Conf. Workshops (NetSoft)*, Seoul, South Korea, Jun. 2016, pp. 116–120, doi: 10.1109/NETSOFT.2016.7502456.

[28] S. Mallon, V. Gramoli, and G. Jourjon, "Are today's SDN controllers ready for primetime?" in *Proc. IEEE 41st Conf. Local Comput. Netw. (LCN)*, Dubai, United Arab Emirates, Nov. 2016, pp. 325–332, doi: 10.1109/LCN.2016.60.

[29] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *Proc. 18th Medit. Electrotech. Conf. (MELECON)*, Lemesos, Cyprus, Apr. 2016, pp. 1–6, doi: 10.1109/MELCON.2016.7495430.
[30] D. Suh, S. Jang, S. Han, S. Pack, M.-S. Kim, T. Kim, and C.-G. Lim, "Toward highly available and scalable software defined networks for service providers," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 100–107, Apr. 2017, doi: 10.1109/MCOM.2017.1600170.
[31] I. Z. Bholebawa and U. D. Dalal, "Performance analysis of SDN/OpenFlow controllers: POX versus floodlight," *Wireless Pers. Commun.*, vol. 98, no. 2, pp. 1679–1699, Jan. 2018, doi: 10.1007/s11277-017-4939-z.
[32] A. Nguyen-Ngoc, S. Raffeck, S. Lange, S. Geissler, T. Zinner, and P. Tran-Gia, "Benchmarking the ONOS controller with OFCProbe," in *Proc. IEEE 7th Int. Conf. Commun. Electron. (ICCE)*, Hue, Vietnam, Jul. 2018, pp. 367–372, doi: 10.1109/CCE.2018.8465731.
[33] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, and M. Guizani, "SDN controllers: A comprehensive analysis and performance evaluation study," *ACM Comput. Surv.*, vol. 53, no. 6, pp. 1–40, Dec. 2020, doi: 10.1145/3421764.
[34] D. Lunagariya and B. Goswami, "A comparative performance analysis of stellar SDN controllers using emulators," in *Proc. Int. Conf. Adv. Electr., Comput., Commun. Sustain. Technol. (ICAECT)*, Bhilai, India, Feb. 2021, pp. 1–9, doi: 10.1109/ICAECT49130.2021.9392391.
[35] S. Bhardwaj and S. N. Panda, "Performance evaluation using RYU SDN controller in software-defined networking environment," *Wireless Pers. Commun.*, vol. 122, no. 1, pp. 701–723, Jan. 2022, doi: 10.1007/s11277-021-08920-3.
[36] *Artificial Conversational Speech*, document ITU-T Rec. P.59, 1993.
[37] Atlassian. *Project Floodlight Documentation*. Accessed: Nov. 21, 2023. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview/
[38] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 19, no. 1, pp. 30–33, Jan. 2015, doi: 10.1109/LCOMM.2014.2371014.
[39] *International Telephone Connections and Circuits—General Recommendations on the Transmission Quality for an Entire International Telephone Connection*, document ITU-T Rec. G.114, 2003.
[40] *Internet Protocol Data Communication Service—IP Packet Transfer and Availability Performance Parameters*, document ITU-T Rec. Y.1540, 2007.
[41] *Network Performance Objectives for IP-Based Services*, document ITU-T Rec. Y.1541, 2011.
[42] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on SDN network utilization," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Toronto, ON, Canada, Apr. 2014, pp. 1734–1742, doi: 10.1109/INFOCOM.2014.6848111.
[43] A. Pekar, J. Mocnej, W. K. G. Seah, and I. Zolotova, "Application domain-based overview of IoT network traffic characteristics," *ACM Comput. Surv.*, vol. 53, no. 4, pp. 1–33, Jul. 2020, doi: 10.1145/3399669.

SYLWESTER KACZMAREK received the M.Sc. degree (Hons.) in electronics engineering and the Ph.D. (Hons.) and D.Sc. degrees in switching and teletraffic science from the Gdańsk University of Technology (GUT), Poland, in 1972, 1981, and 1994, respectively.

From 1972 to 1981, he was an Assistant Professor with the Faculty of Electronics, Telecommunications and Informatics (FETI), GUT. From 1981 to 1994, he was an Assistant Professor, and from 1994 to 2005, he was an Associate Professor. Since 2005, he has been a Professor with FETI. From 2007 to 2018, he was the Head of the Teleinformation Networks Department, FETI. His research interests include IP QoS, GMPLS, ASON and SDN networks, switching, QoS routing, teletraffic, and multimedia services. His current research interests include the developing and applicability of VoIP, IMS/NGN, and SDN technologies. So far, he has published two monographs, more than 270 articles. Among his promoted students, there are 294 engineers and M.Scs., and seven Ph.Ds. He participates and directs projects and research-development grants. Among his distinctions, there is also the TPC membership of conferences, such as ITNAC, KRiT, and SoftCOM.

JACEK ANDRZEJ LITKA received the M.Sc. degree in electronics and telecommunications from the Faculty of Electronics, Telecommunications and Informatics (FETI), Gdańsk University of Technology (GUT), in 2014.

Since 2018, he has been an Assistant with the Department of Teleinformation Networks, FETI. He teaches classes focused on different subject of telecommunication technologies, including principles of transmission, network architectures, digital signal processing, and the security of information systems. His research interest concentrates on the subject of network performance and software defined networks, which are directly connected with his works on a dissertation in the field of information and telecommunication technology. Aside from didactic and research work, he has experience in translating English informatics handbooks, he is an active participant in department's research projects and spearheads department's modernization processes.