
Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2020 SCPE. Volume 21, Issue 2, pp. 337–346, DOI 10.12694:/scpe.v21i2.1698

AN ANALYTICAL MODEL OF A CORPORATE SOFTWARE-CONTROLLED NETWORK SWITCH

VALERY P. MOCHALOV, GENNADY I. LINEST, NATALYA YU. BRATCHENKO, AND SVETLANA V. GOVOROVA*

Abstract. Implementing the almost limitless possibilities of a software-defined network requires additional study of its infrastructure level and assessment of the telecommunications aspect. The aim of this study is to develop an analytical model for analyzing the main quality indicators of modern network switches. Based on the general theory of queuing systems and networks, generated functions and Laplace-Stieltjes transforms, a three-phase model of a network switch was developed. Given that, in this case, the relationship between processing steps is not significant, quality indicators were obtained by taking into account the parameters of single-phase networks. This research identified the dependencies of service latency and service time of incoming network packets on load, as well as equations for finding the volume of a switch's buffer memory with an acceptable probability for message loss.

Key words: switch architecture; thread table; service time; wait time.

AMS subject classifications. 68M10

1. Introduction. Problems and limitations in modern computer networks have led to the development and construction of software-defined networks (SDN - Software Defined Networks). The main approaches of the SDN concept are presented in the guidelines of the International Telecommunication Union - ITU-T Y.3000 series [19], which require the separation of data transmission and management processes, a logically centralized level of control, the use of a unified OpenFlow interface, and virtualization of physical resources. Unlike traditional switching and routing methods based on IP and MAC addresses, the OpenFlow protocol is able to implement more than forty criteria for selecting transmission routes for network packets [11, 18]. In [13], SDN and network functions virtualization technologies (NFV) were investigated with the aim of developing faster and more flexible fault-tolerant solutions. In addition, there is also a wide variety of implementations of the OpenFlow protocol from switch providers [5] mainly related to the construction of additional software layers on existing products. This is of interest for the future development of a general performance analysis and for comparing various SDN solutions (consisting of controllers, switches, and application modules) for more complex scenarios, such as data centers and cloud computing, distributed across wide area networks. [1] describes the task of setting up the manager to work as an SDN controller. The article includes a brief overview of the various OpenFlow SDN-based controllers available in various programmable languages. It focuses on two controllers that support OpenFlow, namely the POX, a Python-based controller, and the Java-based Floodlight controller. The performance comparison of both controllers is tested on various network topologies by analyzing network bandwidth and round-trip latency using an effective network simulator called Mininet. Single, linear, tree-like and user (user-defined) topologies are developed in Mininet by activating external controllers. It should be noted that SDN is a new network architecture that is adaptive, dynamic, efficient, and manageable. The typical architecture of an SDN network [8, 19] is shown in cf. Fig. 1.1. The SDN controller and network operating system periodically update their internal data about the status of network elements, topology, data transfer routes, threads, and resources. Upon receiving a service request, the controller processes the first network packet of the corresponding thread and sets the control and forwarding rules for all subsequent packets, i.e. data management occurs at the thread level. The first packet of each new thread is sent to the controller,

*Department of Information Communications Institute of Information Technologies and Telecommunications North Caucasian Federal University 1 Pushkin Str., Stavropol, 355017, Russia. Corresponding author: Natalya Yu. Bratchenko (nbratchenko@ncfu.ru). This study was carried out with the financial support of the Russian Foundation for Basic Research (RFBR) as part of research project No. 19-07-00856\19.

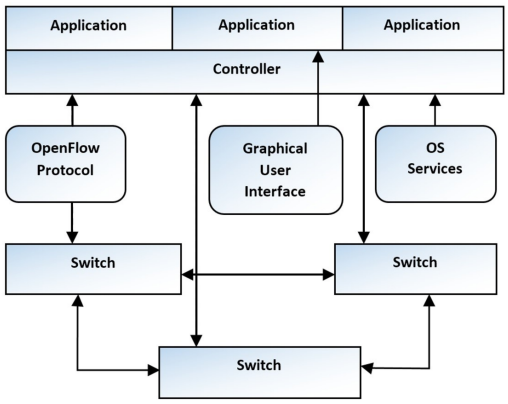

Fig. 1.1. *Typical SDN Network Architecture*

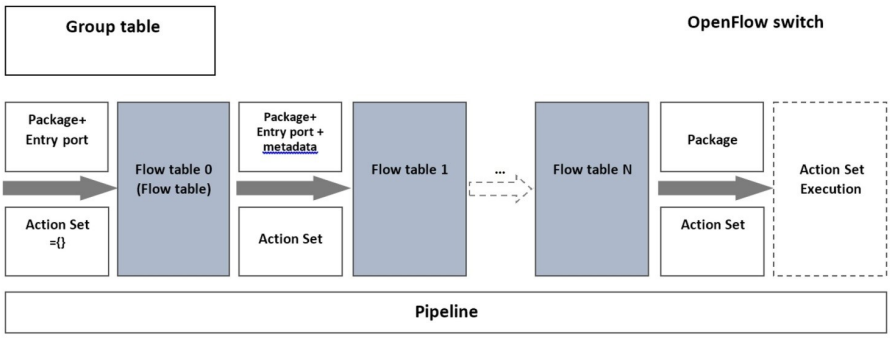

Fig. 1.2. *Contents of SDN Switch Address Tables*

which creates the corresponding entry in the transfer table in the switch. Each switch fills its addressing tables only according to the controller. If we assume that there is no queue of serviced network packets at the controller input, we can describe it as a queuing system (QS) with an infinite number of serviced devices, and the SDN switch model, which includes the phases of receiving and processing packet headers, thread control, and making changes to addressing tables, can be represented by multiphase QS with unlimited memory. The network's operation in stationary mode, as well as its transitions to various states, can be described by the Markov chain [4, 8, 12].

The switch management port is connected to the controller processor by a secure OpenFlow messaging channel. In this case, both a special control network and the existing transport network can be used. Each switch includes a chain of tables connected in series for addressing packet threads (cf. Fig. 1.2), which contain algorithms and instructions for redistributing packets: forwarding to the next table number, to one of the output ports, or to the control input of the controller. Upon receipt of input packets, the address of a received packet is checked against the entries in the thread tables. If an address match is not established, the packet is sent to the controller, which determines the rules for processing it and sets them in the switch's addressing tables. At the same time, the controller, depending on the state of the network (topology, load, deadlocks and packet thread blocks), can change the contents of address tables, and also compares information about the condition of elements.

2. Methods. Each switch contains a set of thread entries that include match fields, counters, and instructions for defined actions related to standard pre-processing and packet forwarding. The structure of the thread entries is shown in cf. Fig. 2.1.

Operations on packet threads can be divided into stages: the switch receives packets from subscribers,

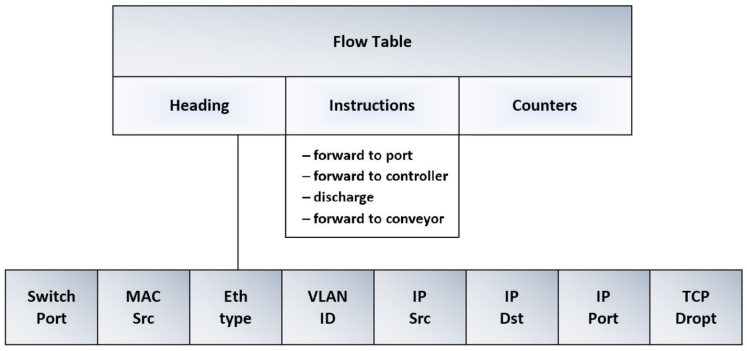

Fig. 2.1. *Switch Thread Entries Structure*

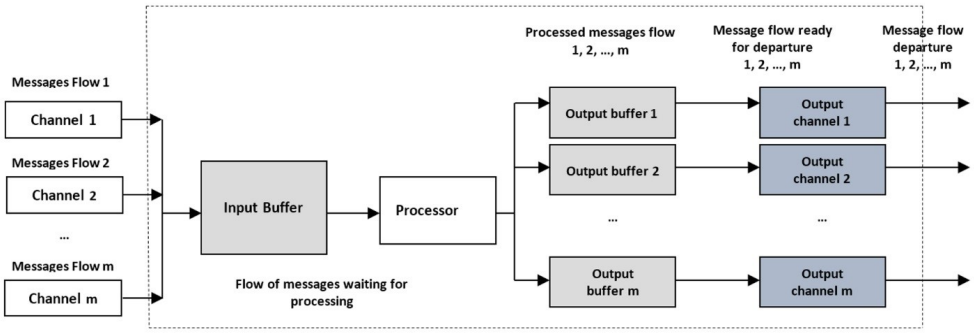

Fig. 2.2. *Switch Architecture Variant*

checks for records of incoming packet threads in the address tables, and forwards requests to the controller, which makes the decision of how to process the packets. A variant of switch architecture is shown in cf. Fig. 2.2.

Packet processing includes the following actions [8, 15, 17]:

1. The packet of the newly formed thread arrives with the speed of the communication channel at the input port of the switch and is placed in its buffer memory.
2. The formation of packet threads. Checking that the address of the received packet matches the entries in the thread tables. If a match is found, then step 5 is executed.
3. If no match is found, the packet is sent to the SDN controller.
4. According to the routing algorithm, the SDN controller adds the corresponding record to the switch and other switches along the transmission path of the thread.
5. Waiting for the release of the channel in the direction of the outgoing port. Packet encapsulation and transmission to the specified address.

Each fragment of the SDN network contains several switches interconnected by high-speed duplex communication channels. We believe that asynchronous sealing should be used on channels connecting subscribers to the switch, as it does not significantly affect overall network performance. Meanwhile, if a packet is sent to the output port of a switch's high-speed channel immediately after arrival, we assume that its service time is zero. If the packet remains in the switch's memory and its header is sent to the controller, then we describe the service time by an arbitrary distribution function. In real network elements, the amount of memory is always limited, so when the established buffer size is exceeded, there is a high probability of packet loss. Buffer memory is a shared resource for all communication channels. The required amount of memory can be estimated based on the given probability of packet failures entering the node. For approximate estimates of the memory size of a switch, we will use functions that account for the volume of requests, the message service time, and the probability of losses. In this case, the number of servicing devices is equal to the number of places in

the drive m and there is no queue. The memory can be represented by the total of m independent single-line QS and failures, service blocking and discipline, distributed according to exponential law. The number of service elements of such a multi-line center is equal to the number of packet threads received at the switch input. The OF-CONFIG protocol used in SDN allows switch resources to be allocated, forming several virtual switches from one physical switch, while distributing physical memory between threads. Message volumes are distributed according to the exponential law $L(x) = 1 - e^{-fx}$. Messages are received on m channels at the intensities $a_i (i = \overline{1,m})$. The thread of received messages is simplest at the intensity $\lambda = \sum_{i=1}^{m} a_i$. The paper [10] reflects clear expressions for determining memory volume with a reasonable likelihood for message refusal. For LMS $M|G|1|\infty$ the Laplace-Stieltjes transform (LST) was found for the distribution function (DF) of the volume of the serviced message $R(x)$:

$$_i(s) = 1 - \frac{a_i}{g_i}\left[\frac{1}{f} + \frac{f}{(s+f)^2}\right] \tag{2.1}$$

where i is the number of the receiving channel; a_i is the intensity of the input thread $(i = \overline{1,m})$; $g_i > 0$ is the message receipt time on channel i; f is the DF parameter of the message volume.

From here, the first two instances of request volume will be equal to:

$$_{1i} = \frac{2a_i}{g_i f^2} = \frac{2\rho_i}{f}, l_{2i} = \frac{2a_i}{g_i f^3} = \frac{6\rho_i}{f^2} \tag{2.2}$$

where ρ_i – is the load of channel i.

$$(_{2i} - l_{1i}^2) = \frac{2\rho_i}{f^2}(3 - 2\rho_i)$$

Then the LST of the stationary total message volume will be equal to:

$$\delta(s) = \prod_{i=1}^{m}\left\{1 - \frac{a_i}{g_i}\left[\frac{1}{f} + \frac{f}{(s+f)^2}\right]\right\} \tag{2.3}$$

From here it follows that the first and second instances of total message volume will be equal to:

$$\delta_1 = \frac{2}{f}\sum_{i=1}^{m}\rho_i, (\delta_2 - \delta_1^2) = \frac{2}{f^2}\sum_{i=1}^{m}\rho_i(3 - 2\rho_i) \tag{2.4}$$

The calculation of the memory volume V is carried out taking into account the probability of message loss:

$$p_{\Pi} = 1 - R(V) \tag{2.5}$$

where $R(V) = \int_0^V D(V-x)dL(x)$; $L(x) = 1 - e^{-fx}$ - DF message volume; $D(x) = \rho(\delta < x)$ - DF stationary total message volume δ

The first two instances of DF $R(x)$ are equal to:

$$r_1 = \delta_1 + \phi_1, r_2 = \phi_2 + 2\phi_1\delta_1 + \delta_2 \tag{2.6}$$

where $\phi_1 = \frac{1}{f}$ - average message size; δ_1, δ_2 - instances of total message volume.

To solve cf. Eq. 2.5 we can use formula:

$$D(V) = p_0 + (1 - p_0)\frac{\gamma(p, gx)}{\Gamma(p)} \tag{2.7}$$

where p_0 is the probability of not receiving requests; $\gamma(p, gx)$ is the gamma distributions are expressed by $\gamma(p, gx) = \int_0^{gx} t^{p-1}e^{-t}dt, \Gamma(p) = \gamma(p, \infty)$, and their parameters p and g are defined as

$$p = \frac{r_1^2}{r_2 - r_1^2}, g = \frac{r_1}{r_2 - r_1^2} \tag{2.8}$$

The next stage of request processing is analyzing the service part of the packets and forming network packet threads. The following actions are performed with each packet [3, 10]:

1. Identifying the package and implementing the procedure for determining its place in the thread.
2. Perform a search on all types of threads.
3. If the thread is not found, then the corresponding packet is transmitted to the controller and a new thread is formed with information about the current packet.

We believe that the request is instantly serviced upon identification of the packet, otherwise, i.e. in the absence of identification, the latency is described by an exponential distribution with parameter p.

The DF of service time for the input packet thread looks like this: $B(t) = p + (1-p)(1-e^{pt})$, and its LST looks like:

$$\beta(q) = p + \frac{(1-p)p}{p+q} = \frac{p(1+q)}{(p+q)} \tag{2.9}$$

from which we get the average service time value:

$$\beta_1 = -\beta'(0) = \frac{1-p}{p} \tag{2.10}$$

If the system load is determined as $\rho = \alpha\beta_1 = \alpha(1-p)/p$, then the LST service timeout is defined as:

$$W(q) = \frac{(1-\rho)(p+q)}{p+q-a(1-p)} = \frac{(1-p)(p+q)}{q+p(1-\rho)} \tag{2.11}$$

The average timeout value is $W_1 = -W'(0) = \frac{p}{p(1-\rho)}$.

If the image takes the form of the rational fraction $\frac{A_n(p)}{B_n(p)}$, and $P_1, P_2, ..., P_n$ - are roots of multiplicity $r_1, r_2, ..., r_n$, so that

$$r_1 + r_2 + ... + r_n = m, \tag{2.12}$$

$$B_m(p) = \beta_0(p-P_1)^{r_1}(p-P_2)^{r_2}...(p-P_n)^{r_n}. \tag{2.13}$$

Then the original can be found with the formula:

$$f(t) = \sum Res\left[\frac{A_n(p)e^{pt}}{B_m(p)}\right] \tag{2.14}$$

If the roots of the denominator $P1, P2..., Pm$ are simple, then

$$f(t) = \sum \frac{A_n(P_k)}{B_m(P_k)} e^p k^t \tag{2.15}$$

Then, the inverse of the LST function $W(q)$ will be determined by the relation:

$$W(t) = \sum Res\left[\frac{(1-\rho)(\rho+q)}{q(q+p(1-\rho))}e^{qt}\right] \tag{2.16}$$

The distribution function of the random variable (RV) U takes the form of:

$$U(t) = p\{U < t\} = \int_0^t W(t-u)dB(u) = \int_0^t e^{-\mu u}W(t-u)dU \tag{2.17}$$

where $B(t) = 1 - e^{-\mu t}$

For the case $p\{W > 0\} = 1 - W(0) = \frac{(n\rho)^n p_0}{n!(1-\rho)}$ the average value of stationary latency will be equal to:

$$W_1 = EW = \int_0^\infty dW(t) = \frac{n^{n-2}\rho^n p_0}{\mu(1-\rho)^2(n-1)!} \tag{2.18}$$

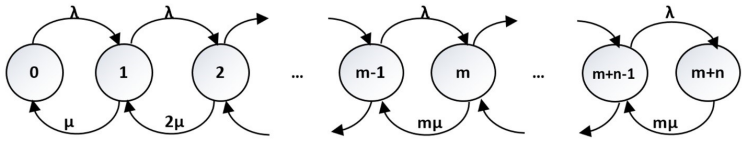

FIG. 2.3. *State and Transition Graph of QS type $M|M|m|n$*

The average value of the stationary service time will be equal to:

$$U_1 = EU = \int_0^\infty t dU(t) = \beta_1 + T_1 = \frac{1}{\mu} + \frac{n^{n-2}\rho^n p_0}{\mu(1-\rho)^2(n-1)!} \tag{2.19}$$

where EW, EU are the mathematical expectations of RV W and U.

To simplify the calculations, the obtained random variables were approximated by the DF $Z(x) = p_0 + (1-p_0)\frac{\gamma(p,gx)}{\Gamma(p)}$, where p_0 is the stationary probability of the absence of requests

$$p = \frac{\delta_1^2}{(1-p_0)\delta_2 - \delta_1^2}, g = \frac{(1-p_0)\delta_1}{(1-p_0)\delta_2 - \delta_1^2} \tag{2.20}$$

where p and g are the gamma distribution parameters, and $\delta_1, \delta_2 - \delta_1^2$, are the first instance and the variance of the total message volume.

At the data transfer stage, the switch executes the instructions associated with a packet, adds routing information received from the SDN controller, and transfers the generated packet thread to the outgoing port to be sent to the communication channel [15, 19]. The methods of transmitting network packets, which determine both the speed of data transmission over communication channels and the storage time of copies of messages in buffer memory, significantly depend on the linear protocols used, the characteristics of acknowledgment stages and *time-out*. We assume that copies of the transmitted packets are stored after the end of their transmission in the buffer memory for some time-out period until confirmation of the delivery of the packets is received. No confirmation necessitates retransmission of the packet. This eliminates possible errors and packet loss due to insufficient buffer memory in the receiving switch. Obviously, the transmission phase of network packets leaving the switch can be described by a system in the form $M|M|m|n$ that is m single-line QS with n place buffers, the simplest incoming thread and exponential distribution of service time. A diagram of the intensity of QS type $M|M|m|n$ transitions is shown in cf. Fig. 2.3.

We know [6, 17] that the probability of finding an $M|M|m|n$ system in a p_k state is:

$$p_k = \frac{\frac{\rho^k}{k!}}{\sum\limits_{k=0}^{m} \frac{\rho^k}{k!} + \frac{\rho^{m+1}}{m \cdot m!} - \frac{1-\left(\frac{\rho}{m}\right)^n}{1-\frac{\rho}{m}}}, 0 \le k \le m \tag{2.21}$$

where $\rho = \frac{\lambda}{\mu}$ is the load.

Accordingly, the probability of a p_{m+s} state is:

$$p_{m+s} = \frac{\frac{\rho^m}{m!}\left(\frac{\rho}{m}\right)^s}{\sum\limits_{k=0}^{m} \frac{\rho^k}{k!} + \frac{\rho^{m+1}}{m \cdot m!} - \frac{1-\left(\frac{\rho}{m}\right)^n}{1-\frac{\rho}{m}}}, 1 \le \rho \le n \tag{2.22}$$

We believe that due to the lack of a free buffer in the receiving switch, a transmitted packet has probability p of being lost. Then the intensity of the output thread is determined as $y = \lambda - \lambda p_{m+n} = \lambda \sum_{i=0}^{m+n-1} p_i$.

The corresponding probabilities of the states of the system in question will be equal to:

$$p_0 = \frac{\rho}{\sum\limits_{k=0}^{m} \frac{\rho^k}{k!} + \frac{\rho^{m+1}}{m \cdot m!} - \frac{1-\left(\frac{\rho}{m}\right)^n}{1-\frac{\rho}{m}}} \tag{2.23}$$

TABLE 3.1
Simulation Results

V	p_{loss}		
	$\rho = 0.4$	$\rho = 0.5$	$\rho = 0.7$
20	0.093	0.171	0.279
30	0.017	0.083	0.073
40	0.0063	0.0038	0.017

$$p_{m+n} = \frac{\frac{\rho^m}{m!}\left(\frac{\rho}{m}\right)^n}{\sum\limits_{k=0}^{m}\frac{\rho^k}{k!} + \frac{\rho^{m+1}}{m\cdot m!} - \frac{1-\left(\frac{\rho}{m}\right)^n}{1-\frac{\rho}{m}}} = \frac{\frac{\rho^m}{m!}\left(\frac{\rho}{m}\right)^n}{\rho}p_0 \tag{2.24}$$

Taking into account lost packets, the output thread intensity will be equal to:

$$y = \lambda(1 - p_{m+n}) = \lambda\left(1 - \frac{\frac{\rho^m}{m!}\left(\frac{\rho}{m}\right)^n}{\rho}p_0\right) = \lambda\frac{\rho - \frac{\rho^m}{m!}\left(\frac{\rho}{m}\right)^n}{\rho}p_0 \tag{2.25}$$

In addition to these characteristics, one can evaluate the quality indicators of the transmitting part of the switch by such parameters as:

- the time the packet stays in the system

$$V = \frac{N}{m\mu(1-p_0)} = \frac{\sum\limits_{k=0}^{m+n} kp_k}{m\mu(1-p_0)}; \tag{2.26}$$

- latency for packets in the queue

$$W = \frac{N_0}{m\mu(1-p_0)} = \frac{\sum\limits_{k=m+1}^{m+n}(k-m)p_k}{m\mu(1-p_0)}; \tag{2.27}$$

- service time

$$\begin{aligned} T_s = V - W &= \frac{\sum\limits_{k=0}^{m+n} kp_k}{m\mu(1-p_0)} - \frac{\sum\limits_{k=m+1}^{m+n}(k-m)p_k}{m\mu(1-p_0)} \\ &= \frac{1}{m\mu} + \frac{(m-1)\sum\limits_{k=m}^{m+n} p_k + \sum\limits_{k=2}^{m-1}(k-1)p_k}{m\mu(1-p_0)} \end{aligned} \tag{2.28}$$

3. Results. Simulation results of the required memory volume V of the switch for a given probability of message loss (p_{loss}) are presented in cf. Table 3.

Cf. Table 3 shows that with an increase in buffer memory, the probability of message loss decreases. With a load of $\rho = 0.4$, an increase in memory volume by a factor of two decreases the probability of information loss by about 14 times, at $\rho = 0.5$ it is 45 times, and at $\rho = 0.7$ it is 16 times. Thus, we can conclude that there is an optimal load for the network, which will minimize losses.

The results of analytical modeling of the formation of packet threads using cf. Eqs. 2.18 and 2.19 are shown in cf. Figs. 3.1 and 3.2.

The presented graphs determine the dependence of the dynamic characteristics of the switch on the load for two message volumes: $\delta_1 = 31.33 \cdot 10^3$ ch, (curve 1) and $\delta_1 = 53.31 \cdot 10^3$ ch (curve 2), with variance $\delta_2 - \delta_1^2 = 105.90 \cdot 10^6$ ch^2. cf. Fig. 3.1 shows that the latency for the message volumes being studied does not significantly differ from load values of 0.6 to 0.7. Then, with an increase in load, latency increases inversely

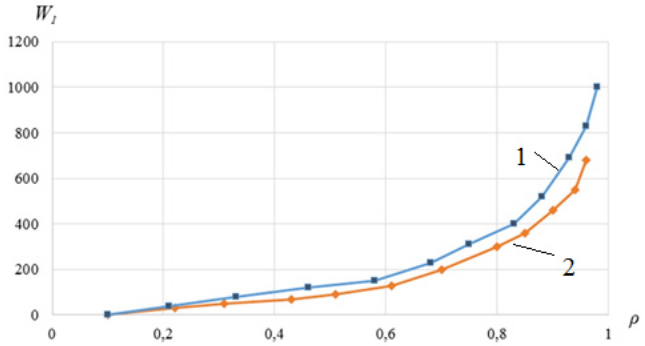

FIG. 3.1. *Average Load Latency*

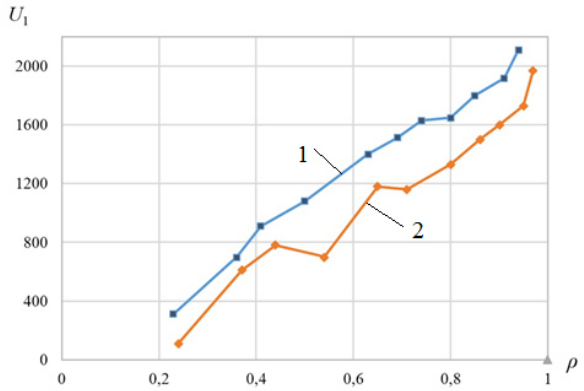

FIG. 3.2. *Relation Between Average Service Time and Load*

with an increase in message size. In general, the functions have a power-law nature, starting from a certain load value (in our case 0.7), latency sharply increases. A longer average latency is observed for a smaller message size. This is done to equalize the processing time of the message packet as a whole. More latency means less time for processing data and, conversely, less latency means more processing time.

Fig. 3.2 shows that the average service time is also inversely related to message size. In contrast to cf. Fig. 3.1, a direct proportional dependence on the load is observed here. In the case of $\delta_1 = 31.33 \cdot 10^3$ ch (curve 1) a certain failure is observed, i.e. a reduction in service time with a load value of 0.5. For the case $\delta_1 = 53.31 \cdot 10^3$ ch (curve 2), the failure is less pronounced and corresponds to a load value of 0.7. Therefore, we can conclude that an optimal relationship exists between message size and service time that can be calculated using cf. Eqs. 2.18 and 2.19.

4. Discussion. Article [14] presents an analytical model of a software-controlled switch on an SDN network and focuses on a study of its stability in the process of changing information interactions. The complexity of the mathematical apparatus, as well as a large number of limitations, are the constraining factors of its application in practice. Evaluations of the switch operation process presented here are time dependent. We know that, when considering random stationary processes, when the characteristics of the system do not change over time, the system can be considered stationary in its established mode, with parameters that do not depend on time. This factor can greatly simplify the model.

Article [16] establishes a connection between message delays in a network switch and the amount of buffer memory. It explores the effect of the internal buffer in software and hardware SDN switches on the system's quality indicators. The disadvantages of this model are the assumptions about the discrete nature of message processing, as well as the possibility of conducting research only at the level of private distributions, which

complicates its application in the analysis of real information systems.

The analytical model of a network switch proposed in [9] provides an efficient enough study of its latencies with various methods of information exposure but does not provide an assessment of quality indicators.

A similar mathematical model is presented in [2]. Limitations on the degree of load used on network devices here do not facilitate conducting research with a fairly wide range of load changes and nonstationary threads.

The analytical model developed for studying the quality indicators of a program-controlled network SDN switch is formally presented as a mass service network (CeMO) with a Poisson thread of incoming packets, exponential service, failures, and locks. We know that when studying this CeMO, it is possible to independently investigate its constituent nodes, which are an $M|M|m|n$ QS type. Therefore, the operations performed by the switch are conventionally divided into three groups: receiving and placing network packets in the buffer memory, processing and generating packet threads, and transmitting network packets to communication channels. A function generator and Laplace-Stieltjes transforms were used for this. The first stage of servicing requests includes the process of writing packets to the switch's multi-line buffer memory. The number of single-line memory elements here corresponds to the number of buffer memory locations; there is no queue. In the next center, the addresses of the received packets are checked against the entries in the thread tables. If a match is found, then a packet thread is generated; if no match is found, the packet is sent to the SDN controller. The third single-line center, which implements the process of transmitting packets leaving the switch, is described as an $M|M|m|n$ system i.e. m independent single-line QS with n place buffers. Considering that, in this case, the relationship between the processing steps is not significant, the packet processing steps being discussed can be considered independent, which is why the quality indicators for this CeMO are obtained by taking into account the parameters of single-phase QS.

5. Conclusion. The new dynamic networking architecture of SDNs has successfully transformed traditional networks into diverse application platforms. However, current SDN technology must be perfected, including with the standardization and unification of various interfaces, ensuring heterogeneous network compatibility, coordinating several controllers, and fixing some security issues. Extending SDN applications also faces major challenges. The use of SDN is no longer limited to campus networks and WANs between data centers but can also be used in wider spaces, including not only terrestrial networks, but also in space. This study developed and investigated a mathematical model for the operation of a three-phase network switch based on the general theory of queuing networks using the Laplace-Stieltjes transform. A number of assumptions were made that made it possible to consider a three-phase switch as single-phase with three independent threads. Using the SDN model presented in this article, network administrators and schedulers can better predict likely performance changes resulting from traffic changes. This will allow them to make operational decisions to prevent small problems from turning into major bottlenecks. The results of the study can be used in the design and operation of communication networks that implement the concept of SDN.

REFERENCES

[1] I. Z. Bholebawa and U. D. Dalal, *Performance analysis of SDN/OpenFlow controllers: POX versus floodlight.*, Wireless Personal Communications, 98(2), 2018, 1679-1699. doi:10.1007/s11277-017-4939-z

[2] Y. Goto, B. Ng, W. K. G. Seah and Y. Takahashi, *Queueing analysis of software defined network with realistic OpenFlow-based switch model*, Computer Networks, 164, 2019, 106892. doi:10.1016/j.comnet.2019.106892.

[3] D. Gross, J. F. Shortie, J. M. Thompson and C. M. Harris, *Fundamentals of queueing theory (1st ed.)*, Hoboken, NJ, USA: Wiley, 2008, doi:10.1002/9781118625651

[4] L. Hanhua, W. Yashi, M. Lijuan and H. Zhenqi, *OSS/BSS Framework based on NGOSS.*, 2009 International Forum on Computer Science-Technology and Applications (pp. 466-471), 2009,. doi:10.1109/IFCSTA.2009.120

[5] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, *Software-defined networking: a comprehensive survey*, Proceedings of the IEEE, 103(1), 14-76, 2015. doi:10.1109/JPROC.2014.2371999

[6] T. Lechler, B. J. Taylor, and B. Klingenberg, *The telecommunications carriers' dilemma: Innovation vs. Network Operation.*, PICMET '07 - 2007 Portland International Conference on Management of Engineering & Technology (pp. 2940-2947), 2007. doi:10.1109/PICMET.2007.4349638

[7] P. N. Brown and Y. Saad, *The problem with threads*, 39(5), 33-42, 2006. doi:10.1109/MC.2006.180

[8] T. Li, J. Chen, and H. Fu, *Application scenarios based on SDN: an overview*, Journal of Physics: Conference Series, 1187(5), 2019, 052067. doi:10.1088/1742-6596/1187/5/052067

[9] S. V. Malakhov, V. N. Tarasov, and I. V. Kartashevskiy, *Teoreticheskoye i eksperimental'noye issledovaniye zaderzhki v programmno-kofiguriruyemykh setyakh. infokommunikatsionnyye tekhnologii [Theoretical and experimental study of delay in software-configurable networks]*, Infokommunikatsionnyye Tekhnologii [Infocommunication Technologies], 13(4), 2015, 409-413. doi:10.18469/ikt.2015.13.4.08 (in Russian)

[10] V. P. Mochalov, N. Yu. Bratchenko, and S. V. Yakovlev, *Analytical model of integration system for program components of distributed object applications*, 2018 International Russian Automation Conference (RusAutoCon) (pp. 1-4), 2018. doi:10.1109/RUSAUTOCON.2018.8501806 (in Russian)

[11] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, *A survey of software-defined networking: past, present, and future of programmable networks*, IEEE Communications Surveys & Tutorials, 16(3), 1617-1634, 2014. doi:10.1109/SURV.2014.012214.00180

[12] M. Olszewski, J. Ansel, and S. Amarasinghe, *Kendo: Efficient deterministic multithreading in software*, ACM SIGPLAN Notices, 44(3), 97, 2014. doi:10.1145/1508284.1508256

[13] N. S. Rao, *Performance comparison of SDN Solutions for Switching Dedicated Long-Haul Connections*, Retrieved from Oak Ridge National Lab. (ORNL), Oak Ridge 2016, TN (United States) website: https://www.osti.gov/biblio/1267045-performance-comparison-sdn-solutions-switching-dedicated-long-haul-connections

[14] K. E. Samouylov, I. A. Shalimov, I. G. Buzhin, and Y. B. Mironov, *Model' funktsionirovaniya telekommunikatsionnogo oborudovaniya programmno-konfiguriruyemykh setey [Model of functioning of telecommunication equipment for software-configurated networks]*, Sovremennyye Informatsionnyye Tekhnologii i IT-Obrazovaniye [Modern Information Technologies and IT-Education], 14(1), 2018. Retrieved from https://cyberleninka.ru/article/n/model-funktsionirovaniya-telekommunikatsionnogo-oborudovaniya-programmno-konfiguriruemyh-setey (in Russian)

[15] J. Simoes, and S. Wahle, *The future of services in next generation networks*. IEEE Potentials, 30(1), 24-29, 2011. doi:10.1109/MPOT.2010.939761

[16] D. Singh, B. Ng, Y.-C. Lai, Y.-D. Lin, and W. K. G. Seah, *Analytical modelling of software and hardware switches with internal buffer in software-defined networks*, Journal of Network and Computer Applications, 136, 22-37, 2019. doi:10.1016/j.jnca.2019.03.006

[17] H. Sutter, and J. Larus, *Software and the concurrency revolution*, Queue, 3(7), 54, 2005. doi:10.1145/1095408.1095421

[18] A. Vishnu Priya, and N. Radhika, *Performance comparison of SDN OpenFlow controllers*, International Journal of Computer Aided Engineering and Technology, 11(4/5), 2019, 467. doi:10.1504/IJCAET.2019.10020284

[19] Y.3300, *Framework of software-defined networking. (n.d.)*, Retrieved from https://www.itu.int/rec/T-REC-Y.3300-201406-I/en

Edited by: Dana Petcu
Received: Feb 16, 2020
Accepted: May 7, 2020