Моделирование и анализ информационных систем. Т. 25, № 2 (2018), с. 207–216
Modeling and Analysis of Information Systems. Vol. 25, No 2 (2018), pp. 207–216

Программно-конфигурируемые сети

DOI: 10.18255/1818-1015-2018-2-207-216

UDC 004.415.25

Organization of Multi-controller Interaction in Software Defined Networks

Morzhov S. V., Alekseev I. V.[1], **Nikitinskiy M. A.**[1]

Received December 25, 2017

Abstract. *Software Defined Networking* (SDN) is a promising paradigm for network management. It is a centralized network intelligence on a dedicated server, which runs network operating system, and is called SDN controller. It was assumed that such an architecture should have an improved network performance and monitoring. However, the centralized control architecture of the SDNs brings novel challenges to reliability, scalability, fault tolerance and interoperability. These problems are especially acute for large data center networks and can be solved by combining SDN controllers into clusters, called multi-controllers. Multi-controller architecture became very important for SDN-enabled networks nowadays. This paper gives a comprehensive overview of SDN multi-controller architectures. The authors review several most popular distributed controllers in order to indicate their strengths and weaknesses. They also investigate and classify approaches used. This paper explains in details the difference among various types of multi-controller architectures, the distribution method and the communication system. Furthermore, it provides already implemented architectures and some examples of architectures under consideration by describing their design, communication process, and performance results. In this paper, the authors show their own classification of multi-controllers and claim that, despite the existence of undeniable advantages, all reviewed controllers have serious drawbacks, which must be eliminated. These drawbacks hamper the development of multi-controllers and their widespread adoption in corporate networks. In the end, the authors conclude that now it is impossible to find a solution capable to solve all the tasks assigned to it adequately and fully. The article is published in the authors' wording.

Keywords: SDN, software defined network, distributed controller, multi-controller, CoVisor, DISCO, ELASTICON, FlowBrick, FlowVisor, HyperFlow, Kandoo, ONIX, ONOS, ORION

For citation: Morzhov S. V., Alekseev I. V., Nikitinskiy M. A., "Organization of Multi-controller Interaction in Software Defined Networks", *Modeling and Analysis of Information Systems*, **25**:2 (2018), 207–216.

On the authors:
Sergey V. Morzhov, orcid.org/0000-0001-6652-3574, graduate student,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: smorzhov@gmail.com

Igor V. Alekseev, orcid.org/0000-0001-8321-2399, Director of the Internet Center, Ph.D.,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: aiv@yars.free.net

Mikhail A. Nikitinskiy, orcid.org/0000-0001-8830-8613, system analyst, programmer,
A-Real Group, Energiya-Info Inc., 144 Souznaya str., Yaroslavl, 150008, Russia, e-mail: man@a-real.ru

Acknowledgments:
[1]The reported study was funded by RFBR according to the research project № 16-07-01103a.

Introduction

Specialists of the universities of Berkeley and Stanford in 2006 proposed the basic ideas of the SDN. This approach involves the construction of computer networks in such a manner that the control plane is separated from the data plane. The control plane is usually a dedicated server that runs network *operating system* (OS) called controller. Not only academic circles were inspired by the SDN architecture, some crucial commercial companies also became interested in it and formed the *ONF* (Open Networking Foundation) community. This community is developing specifications and standards in the field of SDN. According to ONF, the main ideas of the SDN approach are as follows:

- The control plane is separated from the data plane.
- Unified, vendor independent protocol called OpenFlow is used between planes.
- Control plane is logically centralized, implemented as a network OS, called controller, and runs some network applications.

The controller manages the network infrastructure and data streams in it, while network applications are designed to extend the controller's functionality and implemented on other equipment to facilitate the functioning of the controller. In fact, the controller - the core of the SDN - serves as a place where the logic defined by the network applications is converted into a set of switch configuration rules. Thus, the controller can become both a point of failure of the network and its performance bottleneck. If the controller fails, the operation will fail, leading to a complete or partial loss of network management. In this case, the switching equipment will continue to forward packets according to the current routing rules. However, if one or more switching equipment fails, the network will not be able to continue functioning. Therefore, a very important aspect when implementing the SDN approach for corporate, operators, providers or data centers networks is to ensure trouble-free operation of the controller.

To solve the above problem, the distributed interaction of controllers can serve, which will allow, if one controller fails, to retain control over the switching equipment. The presence of several controllers in the network implies the existence of a mechanism for exchanging control information between these controllers. The multi-controller architecture is a set of controllers working together to provide high performance and scalability. The organization of a multi-controller SDN can be logically or physically centralized, having one rank or hierarchical structure. Works in these areas are carried out within the framework of such projects as CoVisor, DISCO, ELASTICON, FlowBrick, FlowVisor, HyperFlow, Kandoo, ONIX, ONOS and ORION.

1. Distributed controller

1.1. CoVisor

CoVisor [1] is a hypervisor that allows you to use multiple controllers located in the same SDN to manage it. Unlike existing hypervisors, which are focused on splitting the network into subnets, CoVisor allows several controllers to simultaneously work with traffic. In

fact, CoVisor allows you to choose the best functionality from different controllers. For example, you can use the firewall application from the Floodlight controller, the load balancer from Ryu and so on, regardless of the programming language and libraries used. The resulting set of functions can be applied to traffic in whole or in part. CoVisor abstracts from a specific network topology and creates a virtual one in its place, which allows the administrator to regulate controller access to packages that it can monitor, modify and redirect. The main technical feature of CoVisor is a set of efficient algorithms that handle the policies of controllers and transform the virtual network into a set of specific OpenFlow rules, as well as ensuring their updating. The hypervisor prototype is based on the OpenVirteX hypervisor and is written in Java. CoVisor testing results show that it handles packets more quickly than OpenVirteX by several orders of magnitude.

1.2. DISCO

DISCO [2] is a SDN multi-controller. Each DISCO controller monitors its own local SDN and provides interconnection of controllers via unique communication channels. The control and transmission of information in these channels is carried out via the Messenger module using the AMQP protocol. The communication channels use specialized agents (Monitoring agent, Reachability agent, Connectivity agent, Reservation agent and others). Specialized agents and Messenger module are implemented in the inter-domain part of the controller, which is responsible for monitoring, synchronization, data exchange between DISCO controllers. The intra-domain part of the DISCO is based on the Floodlight controller. This part is responsible for the direct operation of the controller over the dedicated SDN or network applications. In their work [2], the authors evaluate experimentally the following: breaking the connection between different local networks and rebuilding the interconnection of controllers, processing traffic by controller m, which has priority from controller n, idle time when the controller is migrated to another virtual machine.

1.3. ELASTICON

ELASTICON [3] is a cluster of standalone controllers that share the workload to provide a global network view for the management layer. This global view of the network uses the Distributed Data Store module, through which a logically centralized controller is created. Each module has a TCP channel that connects it to neighboring controllers. This ensures the exchange of messages between controllers, the switching of responsibility zones or coordinating actions while switching equipment is migration process.

The following example can illustrate the typical use case of ELASTICON. The switching equipment is connected to several controllers. One of them is the master and the others are slaves. Each controller has a core controller module that is responsible for connecting the switching equipment and for interacting this equipment with the controller core applications. It should also be noted that this module is responsible for choosing which controller will be connected to the new switching equipment and for the migration of this equipment between the controllers. The controller core applications are responsible for the executable logic of the switching equipment. The information stored in them is transmitted via the Distributed Data Store module to a similar application

in another controller when the switching equipment is migrated. Balancing of switching equipment between controllers is performed periodically and does not depend on traffic load.

1.4. FlowBricks

This controller is based on the Floodlight kernel. FlowBricks [4] is built in such a way that it acts as an intermediary between various controllers and SDN. This approach was chosen because different parts of different controllers operate more or less efficiently. Thus, FlowBricks can send both sequentially and in parallel a packet-in to the controllers. The processing of responses from controllers can be configured and FlowBricks may decides how to modify the packet-out. The prototype of the controller showed that the processing time of the OpenFlow package has not increased significantly.

1.5. FlowVisor

FlowVisor [5] is an approach to organization of the SDN architecture that involves creating an additional layer for processing OpenFlow packets between the SDN controller and OpenFlow equipment. FlowVisor intercepts, modifies and redirects all packets exchanged between controllers and equipment, so that for each individual controller the network (or part of it) looks controlled only by it. In this way, the network is divided into layers that work independently. On the one hand, the advantage of this approach is the relative independence of a particular SDN controller, since the manipulations are done directly with OpenFlow packages, without affecting the controller's operation. As an analogue, you can consider the use of VLAN in a classical network. On the other hand, the administrator is obliged to configure, control and resolve conflicts between controllers. The system was based on the NOX controller.

1.6. HyperFlow

HyperFlow [6] is a network application for the NOX controller, which allows you to turn this controller into a distributed one. This is achieved through WheelFS [7], which provides passive data synchronization between network applications of controllers. This synchronization build on the publishing/subscribing events method. It ensures that the controller does not duplicate the transmitted information, and minimizes the amount of traffic transmitted by the controllers.

1.7. Kandoo

The Kandoo [8] project involves building a hierarchy of controllers in such a way that the lower level of the hierarchy is the controllers that manage their domains, and the top level is the root controller that controls the lower-level controllers. Low-level controllers process packet-in requests for which they have solutions, otherwise they send it to the top-level controller, which has a complete view of the entire SDN. For the Kandoo project to optimize the operation of the SDN two main applications have been created: detect and reroute. The first application works on the lower level of the controllers. It determines

large flows (large is considered to be a stream of more than 1 MB). They overflow the channels and reduce the efficiency of the SDN to transfer other flows. When one of the controllers determines a large flow, it passes this information to the upper level to rebuild the routing of all threads in the controlled SDN.

1.8. ONIX

ONIX [9] is a distributed controller that can be run on a cluster. At the same time on each node of the cluster - a dedicated server - several Onix controllers can be started. The SDN controlled by ONIX is divided into four logical components: a physical infrastructure that includes all network devices; the connection infrastructure, which is the connection between the physical network and ONIX; the management logic that is presented by the ONIX API for network applications and, finally, the ONIX controller itself, interacting with the physical infrastructure.

The ONIX API allows network applications to read and write the status of any network element in the SDN. Each network element can contain one or more data objects to which network applications should have access. In the NIB information database, each state of the network is stored as a graph. ONIX controllers support NIB synchronization and each network applications can read and write status in NIB.

For scalability, the ONIX controller offers three methods: the dynamic load sharing between multiple copies of the controller, the creation of a hierarchical structure with the representation of aggregation of controllers at the lower level and the ability to transfer the state of applications.

1.9. ONOS

The first prototype of the ONOS SDN controller [10] was based on the Floodlight. It has the switch manager, I/O loop, link discovery, module management, and REST APIs. The main characteristics of this prototype were global network view, scalability and fault tolerance. To ensure higher performance, ONOS was build using the following frameworks. Firstly, it is the Titan – a distributed graph database, which is storing the network topology. Secondly, it is Cassandra – a distributed No-SQL database, which is implemented as a key-value storage and were used for storing information about switches, ports and so on. Thirdly, is Blueprints API – the interface, on which network applications receive information about the current network topology.

If the global network view has changed, OpenFlow managers detect it and implement changes in the real network. Each controller, in this architecture, is responsible for its own domain. In this case, each switch is connected to several controllers. If a controller stops responding, then using the ZooKeeper algorithm, there are elections among the active controllers, who will take responsibility for managing the SDN slice that is left without a controller.

The main drawback of the proposed approach is low performance due to the high overhead required to update the network topology stored in Titian and Cassandra. With the fall of one controller, the reorganization of the SDN lasted up to 30 seconds, which is unacceptable.

While creating the second prototype, the focus was on improving the performance of the first one, keeping the Global network view. In this case, there are two main areas where the performance gain can be got. The first one concerns making remote operations as fast as possible, while the second approach focuses on reducing the number of remote operations. It was decided to change the Titan, Cassandra and Blueprint API stack to faster in-memory DBs. The creators chose RAMCloud DB, because it provides an interface compatible with the Blueprint API. In addition, the authors suggested using topology cache. The essence of this technology lies in the fact that topology changes are not added immediately in the DB, but store in the cache. If after some time there are no any new changes, then the cache data is inserted to the DB. In addition, to improve the inter-controller communication through notifications, the authors applied inter-instance publish-subscribe event notification and communication system based on Hazelcast. With the help of it, several notification queues have been created. Network applications can subscribe to them, and, thus, they will quickly receive the information they need.

1.10. ORION

ORION [11] is a hybrid hierarchical control plane, which is a mix of flat and hierarchical architectures. It tries to combine the benefits of both designs and put them all together in a hybrid structure. A network controlled by ORION has three layers: the physical layer, which contains all the physical devices, such as OpenFlow switches; the bottom layer of the control plane that includes the controllers, which handle collecting physical device and link information, as well as dealing with intra-area requests and updates; and, finally, the upper layer, which contains the domain controllers.

ORION interdomain controllers' communication relies on the Horizontal Communication Module that synchronizes the information among domain controllers to build a global network view. The interarea and domain controllers' communication relies on the Vertical Communication Module, which is a set of TCP connections that permit area controllers to send the abstracted topology of the infrastructure layer and request information from the domain controller when a host in some domain wants to reach a particular host in another domain.

ORION has made a theoretical and an experimental evaluation to test the performance of its control plane. On one hand, the theoretical evaluation shows that the computing time of ORION has a linear growth, which is much lower than the traditional Dijkstra routing algorithm. On the other hand, the experimental evaluation tried to verify the feasibility and the effectiveness of ORION shows that the delay time increases gradually.

2. Distribution systems analysis

After analyzing the presented multi-controllers and the principles of organizing a distributed, scalable and fault-tolerant control system for SDN, we classify the controllers into subgroups and define the methodology for creating the distributed SDN.

Fig. 1 shows the classification of multi-controllers. Physically centralized multi-controllers are monocontrollers with multi-threaded organization of event processing. For examples, NOX and Beacon controllers, as well as the CoVisor, FlowBrick or FlowVisor

hypervisors are typical representatives of this group. Physically distributed multi-controllers are controllers running as separate instances. Logically distributed multi-controller is a cluster of SDN controllers that are responsible for different domains and perform various functions. Depending on the functionality, they can have hierarchical (Kandoo) or flat (ORION) architecture. Logically centralized multi-controllers performs a single task in one domain, which is distributed between them depending on the total load. They can have both a dynamic structure (the number of instances increases and decreases as necessary) – DISCO, ONIX, ONOS, and the static structure (the number of instances is invariable) – Elasticon, HyperFlow. Representatives of this class have flat architecture.

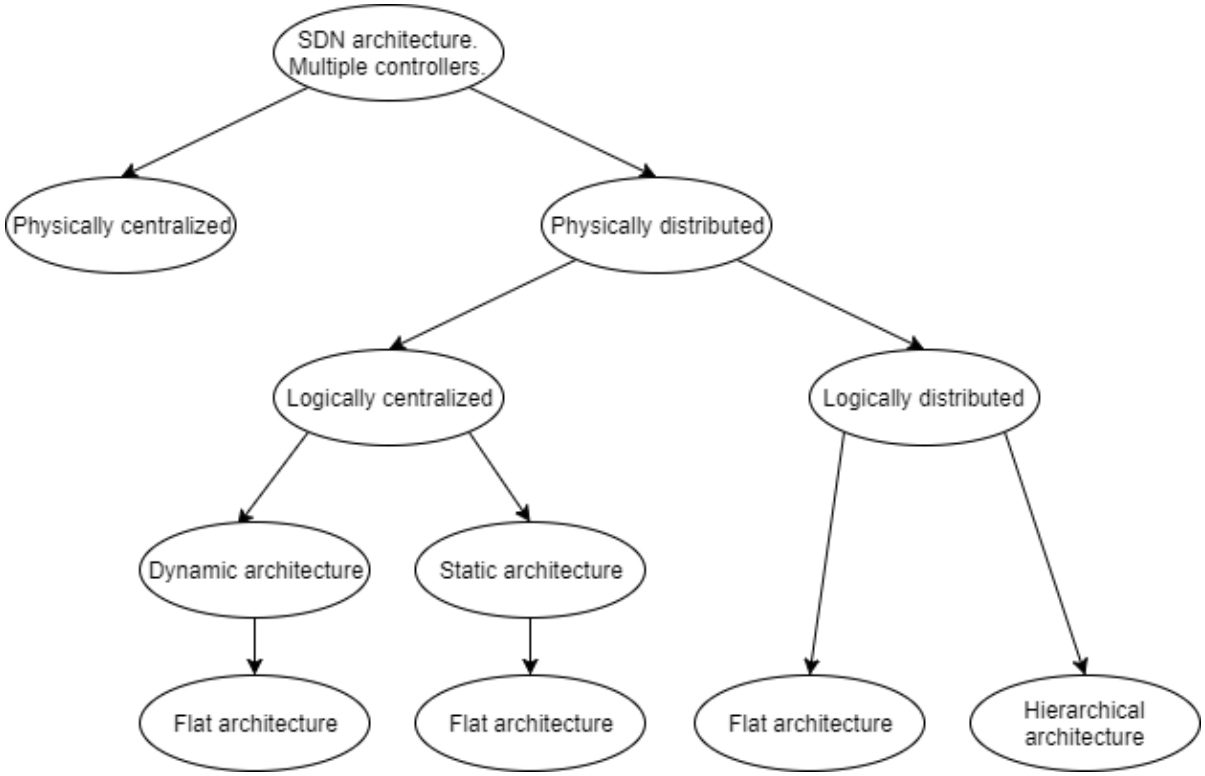

Fig 1. The classification of multi-controllers

According of the above-mentioned review of controllers, the distributed system can be developed with the help of:

- hypervisors – physically centralized controllers;
- special applications functioning as a mediator between the controller and network applications (HyperFlow);
- special controller core modules linked to corresponding module of another controller via TCP channel. Most multi-controllers function this way.

The last approach is the most scalable and fault-tolerant. Developers implement it using publish-subscribe event notification mechanism. Synchronization is achieved by

updating the information on the monitored SDN and the actions performed on it between different controllers.

Despite the fact that the main ideas of the SDN approach are the vendor dependence reduction and centralized network management, the developers of distributed controllers again impose a restriction that involves using only their multi-controller. On the one hand, there are hypervisors that can solve this problem, but they do not solve the bottleneck problem. On the other hand, modern controllers are just a tool for collecting information and installing logic into network applications, while controllers and network applications are controlled by a higher-level entity, for example, the ETSI MANO standard solutions. Thus, to make this system to functioning properly, both highly qualified specialists and equipment of the data center level are required.

A possible solution for creating distributed interaction between different controllers and various network applications, without using a higher level entity capable for managing a distributed SDN that is not a data center, is the creation of a unified kernel module which should follow several requirements. Firstly, the interaction between controllers and network applications must be performed using the RPC method, since many controllers already use it. Secondly, the interconnection between the modules should be performed via an encrypted TCP channel. Thirdly, the module must have different APIs for interaction with various controllers and network applications. In fact, this module should become a distributed hypervisor at the core level of the controller with the ability to interact with network applications.

3. Conclusion

To create a unified kernel module, the authors developed the PreFirewall [12] network application and a random rule generator for the firewall of the Floodlight controller core module. The generator allows you to simulate the interaction of an arbitrary network application with the SDN controller. In addition, the external module of the Floodlight controller core Topology Tracker [13] was developed for storing and exchanging information on the current network topology. It also allows subscribing to events of a similar module with a distributed controller. The authors designed an external module of the Floodlight SDN controller called DEventBus [14]. It is able to send events that occurred in module Topology Tracker to network applications signed for them. The DEventBus module is a prototype for the organization of interaction between multi-controller and network applications.

In the future, the authors are planning to unify this module for organizing multi-controller interaction. For this purpose, we are going to implement various API algorithms for checking the availability of resources, opening and supporting communication channels.

References

[1] X. Jin, J. Gossels, J. Rexford, D. Walker, “CoVisor: A Compositional Hypervisor for Software-Defined Networks”, Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’15) (May 4–6, 2015, Oakland, CA, USA), 87–101.

[2] K. Phemius, M. Bouet, and J. Leguay, “DISCO: distributed multi-domain SDN

controllers", Proceedings of the IEEE Network Operations and Management Symposium (NOMS'14) (Krakow, Poland, May 2014), 1–4.

[3] A. Dixit et al., "Towards an elastic distributed SDN controller", Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'13) (ACM, Hong Kong, 2013), 7–12.

[4] A. Dixit, K. Kogan, P. Eugster, "HyperFlow: a distributed control plane for OpenFlow", Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols (October 21 – 24, 2014), 287–292.

[5] L. Liao, A. Shami, V.C.M. Leung, "Distributed flowvisor: a distributed flowvisor platform for quality of service aware cloud network virtualisation", *IET Networks*, **4**:5 (2015), 270–277.

[6] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow", Proceedings of the Internet Network Management Conference on Research on Enterprise Networking (INM/WREN'10) (Berkeley, Calif, USA, 2010), 1–6.

[7] J. Stribling et al., "Flexible, wide-area storage for distributed systems with WheeIFS", Proceedings of the 6th USENIX symposium on Networked systems design and implementation (Boston, Massachusetts, April 22 – 24, 2009), 43–58.

[8] S.H. Yeganeh, "Kandoo: a framework for efficient and scalable offloading of control applications", Proceedings of the 1st ACM Workshop on Hot Topics in Software Defined Networks (HotSDN'12) (Helsinki, Finland, August 2012), 19–24.

[9] M.T. Koponen et al., "Onix: a distributed control platform for largescale production networks", Proceedings of USENIX Operating Systems Design and Implementation (OSDI'10) (October 4-6, Vancouver, Canada, 2010), 351–364.

[10] U. Krishnaswamy et al., "ONOS: an open source distributed SDN OS", Proceedings of the third workshop on Hot topics in software defined networking (HotSDN'14) (Chicago, Illinois, USA, August 22, 2014), 1–6.

[11] Y. Fu et al., "Orion: a hybrid hierarchical control plane of software-defined networking for large-scale networks", Proceedings of the 22nd IEEE International Conference on Network Protocols (ICNP'14) (Raleigh, NC, USA, October 2014), 569–576.

[12] S. Morzhov, I. Alekseev, M. Nikitinskiy, "Firewall application for Floodlight SDN controller", International Siberian Conference on Control and Communications (SIBCON) (12–14 May, 2016, Moscow, Russia), 1–5.

[13] A.A. Noskov, M.A. Nikitinskiy, I.V. Alekseev, "Development of an active external network topology module for Floodlight software-defined network controller", *Automatic Control and Computer Sciences*, **50**:7 (2016), 546–551.

[14] I. Alekseev, M. Nikitinskiy, "EventBus Module for Distributed OpenFlow Controllers", Proceedings of the 17th Conference of Open Innovations Association FRUCT (20–24 April, 2015, Yaroslavl, Russia), 3–8.

Моржов С. В., Алексеев И. В., Никитинский М. А., "Организация мультиконтроллерного взаимодействия в программно-конфигурируемых сетях", *Моделирование и анализ информационных систем*, **25**:2 (2018), 207–216.

DOI: 10.18255/1818-1015-2018-2-207-216

Аннотация. Программно-конфигурируемая сеть (ПКС) – это перспективная парадигма управления сетью, в которой для повышения производительности уровень управления сетью отделен от уровня передачи данных и реализуется программно на выделенном сервере. Несмотря на очевидные преимущества подхода централизованной архитектуры управления ПКС, она создает новые проблемы, связанные с надежностью, масштабируемостью, отказоустойчивостью и интероперабельностью сети. Эти проблемы встают особенно остро для больших сетей дата-центров

и решаются путем объединения нескольких контроллеров ПКС в кластер, называемый мульти-контроллером. В данной статье представлен обзор некоторых наиболее популярных мульти-контроллеров ПКС, выделены их сильные и слабые стороны, а также приведена классификация используемых ими подходов к организации распределенного взаимодействия. Подробно рассматриваются различия между несколькими типами архитектур мульти-контроллеров, среди которых есть как находящиеся на этапе разработки, так и успешно функционирующие в данное время в дата-центрах. Авторы на примере разработанной ими классификации мульти-контроллеров показывают, что, несмотря на наличие неоспоримых преимуществ, все рассмотренные контроллеры имеют недостатки, которые необходимо устранить. Устранение данных недостатков поможет развитию мульти-контроллеров и сделает возможным их широкое использование в корпоративных сетях. В заключение авторы приходят к выводу, что на данный момент нельзя найти решение, способное в полной мере решить все поставленные задачи. Статья публикуется в авторской редакции.

Ключевые слова: ПКС, программно-конфигурируемая сеть, распределенный контроллер, мульти-контроллер, CoVisor, DISCO, ELASTICON, FlowBrick, FlowVisor, HyperFlow, Kandoo, ONIX, ONOS, ORION

Об авторах:
Моржов Сергей Владимирович, orcid.org/0000-0001-6652-3574, магистрант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: smorzhov@gmail.com

Алексеев Игорь Вадимович, orcid.org/0000-0001-8321-2399, канд. физ.-мат. наук, директор Интернет центра,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: aiv@yars.free.net

Никитинский Михаил Александрович, orcid.org/0000-0001-8830-8613, программист-аналитик,
ООО «Энергия-Инфо», ул. Союзная, 144, г. Ярославль, 150008 Россия, e-mail: man@a-real.ru

Благодарности:
Работа выполнена при финансовой поддержке РФФИ, проект № 16-07-01103а.