EventBus Module for Distributed OpenFlow Controllers

Igor Alekseev
Director of the Internet Center
P.G. Demidov Yaroslavl State University
Yaroslavl, Russia
aiv@yars.free.net

Mikhail Nikitinskiy
System analyst, programmer
A-Real Group, Energiya-Info Inc.
Yaroslavl, Russia
man@a-real.ru

***Abstract*—In this article the authors examine the concept of software defined network (SDN). In the beginning there is a short historical background of the "software defined network" as the scientific and technological concept given, by whom it was introduced and what it means. Substantial attention is paid to the OpenFlow protocol. Further, the authors consider development of SDN technology similarly to ideas suggested in MPLS concept, discuss the issues of distributed controllers using ONIX, Kandoo as examples and describe the mechanism of interaction between applications and SDN controller, which they have devised.**

I. Introduction in software defined network

Initially, global IP-based networks have been built around the concept of an autonomous system (AS) [1]. This concept allows network scaling and expansion through transmission of packets to the next transit area. This network engineering principle is simple and has a proven fault tolerance and scalability. This is how Internet is constructed. The principle of an autonomous system does not allow to move destination points, i.e., change the topology of the network, without changing endpoint identification as it is closely related to the packet delivery process. Topological location of network host, with network interfaces dictates the identification of each interface. In addition, the use of only the basic autonomous system makes it difficult to leverage the other qualities of the communicating node as identification, such as logical grouping, access control, quality of service, interconnection, or certain other aspects that are related to the sequence of packets that form a stream or a network session. To solve these issues Internet Engineering Task Force (IETF) has introduced several additional standards such as virtual LANs (VLAN) and virtual private network (VPN), among many other standards. An increasing number of standards leads to growing complexity of the specification of network elements and the configuration of network interfaces for network operators.

A. *Causes for new paradigm*

Explosive growth and proliferation of mobile devices and the content for them, server virtualization and dissemination of cloud services are the main trends in rethinking traditional network architectures.

Details of the drivers for change in paradigm :

- Changing patterns of traffic in corporate and global networks [2]. Unlike client-server architecture, where the bulk of communication takes place between clients and server, modern applications create multiple streams of data between different computers and not only between servers. In addition, users also changes the traffic patterns using mobile devices and demanding full access to corporate resources and applications from anywhere at any time. Network architecture should easily deal with various mobile devices, such as tablet computers, smart phones, communicators and laptops allowing for precise control, corporate data and intellectual property protection while preserving usage policy and data in the network.
- Cloud services development [3]. Business actively uses public and private cloud services. Business logic drives the need to have the access to applications, infrastructure and other IT services on demand and with high degree of preciseness. Additionally security issues related to private and public clouds are very important. The architecture must provide the possibility of various changes in corporate structure both internal and those caused by external mergers and acquisitions. Thus, following the computing resources network infrastructure must be able to scale elastically, be reorganized on demand and preferably be managed by the same instruments as data storage infrastructure and computing resources.
- Sharp growth in data volumes. Modern applications, like business analytics process huge data amounts and require information exchange between multiple servers, which can have initially random set of links with each other. Growth in data volume on the network and unpredictability of possible routes also places high demand on the network, requiring it to scale more substantially, than traditional network paradigm allows.

B. *Conflict between new requirements and traditional paradigm*

All these factors, which in fact are new functional requirements for the network architecture lead to a paradigm shift, because the existing architectural principles are not able to meet the new requirements [4]. Network technology today is a set of protocols designed to ensure connectivity of hosts on different types of channels in different topologies and distances. These protocols have been developed in isolation from each other (actually this isolation underlies the layered model of network architecture, whether OSI or TCP/IP). The result is increased network complexity. For example, to add a new device you need to change the configuration of switches, routers, firewalls, etc., to update the access control lists, information about VLAN, quality of service and other mechanisms using administration tools that are specific to a particular equipment, and dependent on the equipment manufacturer, the type and version of the software. Thus modern network architecture has difficulty in timely support of changing application requirements and business logic, has insufficient scalability and is difficult to set up and maintain, is dependent on the specific-vendor solutions.

C. *Software defined networking paradigm*

Due to inability of existing network architecture to tackle new functional tasks, which users demand from the network, a new level of software defined network arises. Software defined network is a new approach to network architecture construction, where network control level and data transfer level are separated by moving control functions to software applications, which run on an entity called network controller. The main idea of SDN was put forward by scientists in Stanford and Berkeley in 2006. Their ideas found support not only in research institutions around the world [5], but were also welcomed by over 40 leading network equipment manufacturers and main networking companies, which formed Open Networking Foundation in 2011.

The high degree of interest shown by IT companies is explained by results of practical tests in which SDN allowed to increase the effectiveness of networking equipment in data centers by 25%-30%, decrease network management costs by more than 30%, make network infrastructure management in data centers more flexible, substantially enhance the security and begin easy development of new services with timely equipment updates with new rules. Main ideas in the foundation of SDN are:

- Separated level of data transfer and control,
- Logically centralized management functions are performed by a controller with network operating system (NOS) and a set of network applications installed on top of NOS,
- Single, unified and vendor-independent interface between control level and data transfer level (OpenFlow protocol).

Software defined network consists of switches involved in data transfer (forwarding) and controller, which calculates routes and rules of data transfer. Controller manages a set of switches via protected channels.

According to OpenFlow specification every switch maintains one or more flow tables. Each table in switch contains a set of rules guiding data transfer [6]. Each rule contains match fields (to determine the flow of a packet), counter fields and action fields. OpenFlow switch operates in a relatively simple manner. Every incoming packet has its header (bit sequence of fixed length) analyzed. This bit field is matched to flow table. In case there is a match the packet and its header undergo changes/actions, set in action field of the matched line in flow table. These actions include rewriting of packets header and its packet routing. Actions also guide the counter modification, which can be used to gather network statistics. If header matching is failed, that is packet cannot be related to known flow, such packet is sent to controller, whose responsibility is to inspect the packet and make a decision on new flow and disseminate new flow data into controlled switches. The controller is a physical server with installed network operating system and a set of applications. The controller is the central entity of software defined network, where the basic SDN management functionality is located.

Network operating system provides access of network applications to network management functions and monitors network hardware configuration. Contradictory to traditional notion of network operating system as operating system with network protocol stack, in SDN this term is used to denote system monitoring and controlling resources of the whole network rather than resources of a particular node. At present over 20 realizations of NOS for SDN are known: NOX, POX, Beacon, Maestro, Trema, BigSwitch, FloodLight and others.

Much like traditional operating system NOS provides application programming interface (API) for network management applications [7]. It contains functions for controlling flow tables in switches such as adding, deleting, modifying rules and gathering of various statistics. Thus the actual network management is performed by applications, which use API of NOS. NOS API allows building applications which operate higher level abstractions, e.g. use user name and hostname avoiding use of lower level parameters, such as IP and MAC addresses. Due to this abstraction level management commands are always executed. Maintaining of this abstraction level requires NOS to translate between high level abstractions and lower level technology dependent configurations [8].

II. OpenFlow protocol

Software defined network requires some techniques to enable communication between the control layer and data transfer devices. Currently, the most popular and actively developing is OpenFlow protocol. At the time of this writing there are two branches of the protocol: 1.3.4 at [9] and 1.4.0 at [10].

Like a set of instructions for the processor, OpenFlow defines basic primitives to help third-party application program the logics of data transmission network devices.

The basic idea of the OpenFlow protocol is simple: most of today's Ethernet-switches and routers contain a table of flows that are used to implement firewalls, NAT, QoS, and collect statistics. At the same time, the flow tables of equipment of different manufacturers are different. OpenFlow uses a standard set of features and provides an open protocol for programming flow tables on various switches and routers.

OpenFlow is implemented on both sides of the interface between network devices and the network software controller. OpenFlow uses the concept of flows to identify traffic based on pre-defined rules of comparison, which are set statically or dynamically by the network controller. Thus it is possible to control how traffic passes through the network devices, depending on parameters such as the specific use of the network, load, available resources of cloud and distributed applications, available storage resources. Accordingly, the network is programmed with a granularity of individual flows.

Simple OpenFlow-switch is a data element that forwards packets between ports as defined by the remote controller at Fig. 1.

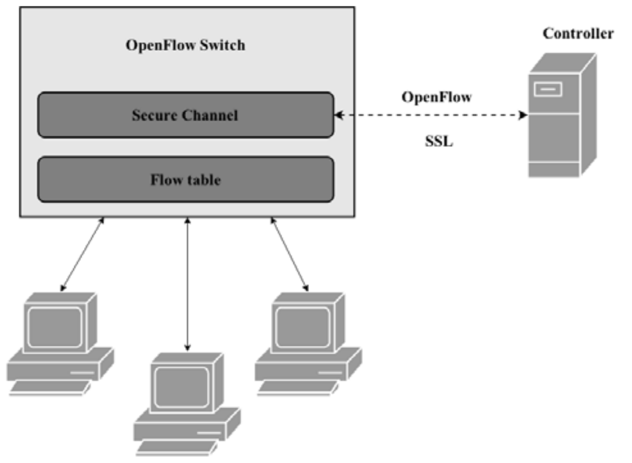

Fig. 1. OpenFlow switch. The table streams controlled by the remote controller through the encrypted channel

OpenFlow switch must contain at least three parts:

- Flow table. It contains data about flows and action associated with each flow that tells the switch how to process this flow.
- Secure channel. It is used to transmit packets and commands between the remote controller and the switch.
- OpenFlow protocol. Provides an open and standard way of interaction between switch and controller.

In this representation, flows can be determined by various features and are limited only by the implementation of flow table. For example, a flow can be TCP session, or all packets from specific MAC or IP addresses, or all packets with the same VLAN tag, or all packets with the same port number on the switch.

As a fast, but not the most effective solution, commercial switches and routers can be modified to work with OpenFlow protocol by adding flow table, a secure channel and OpenFlow protocol support at [11]. Typically, the flow table may be implemented using hardware tables, for example, TCAM (Ternary Content Addressable Memory); secure channel and the protocol can be ported to the operating system of the device.

Other, more effective principles of switch and router arhitecture are also often cited in the literature in relation to SDN approach. For example, in [12] improvements are suggested which can affect the principles of the switch hardware logic and eliminate certain disadvantages of OpenFlow protocol. The disadvantages of the logic of modern switches mainly are that they allow to perform a sequence of match / action operations only over a very limited set of fields. And the OpenFlow protocol greatly limits the range of options for packet processing. To change this situation the authors of [12] propose RMT (reconfigurable lookup table) mechanism, which will change the logic of switch ASIC, changing the characteristics of the fields on which search is performed on the move and offering a wider range of actions on the packets, which matched the search criteria. Moreover, the whole system can operate at terabit speeds without loss of performance.

III. Direction of recent SDN development, using MPLS concepts

In [13], the authors propose to return to the MPLS technology and use it to further develop solutions SDN. Assuming that the three most important requirements for the infrastructure of the modern networks are: simplicity of design/logic equipment operation, which allows to increase the processing speed, independence of the model/brand of equipment, and readiness for a change, instead of need to replace equipment while developing network technology. The software component of networks has only one required characteristic — maximum flexibility and programmability.

We consider the network as a system of interfaces, where interface is the point in which the control information is transmitted between entities of the network infrastructure. From this point of view, there are three interfaces:

- Node-network in which end data sources inform the network about their needs.
- Operator-network, in which network operators (service providers or administrators of corporate networks) inform the network about their needs.
- Packet-switch, determining how the packet is identified to switch (or more generally any network device that performs packet forwarding).

In the traditional model of the Internet network simply moves packets from source to destination. Each router decides independently for each packet. Therefore interfaces node-network and a packet-switch are identical, and the operator interface network is not formalized.

In network technology MPLS, which offers a clear difference between the boundary network and its kernel, there is a difference between the node-network and packet-switch interfaces. Whereas in the first case this represents an interface IP packet header, and in the second case a special mark, which by means of the traffic is marked on the boundary router. However, the technology MPLS is not formalized interface network-operator.

In the concept of SDN operator-network interface underwent formalization, while the other two interface remained unchanged comparing to the traditional architecture of the network. OpenFlow protocol enables the controller to manipulate the switching/routing equipment, using standard Ethernet header components, IP, transport layer protocols, causing sending the packet to a specific port, or modification of these fields. That is the need to interpret packet header by network devices remains SDN problem. This in turn does not allow to reach high speeds of packet processing in hardware, binds the network to specific protocol for example the transition from

IPv4 to IPv6 will raise the need to consider other header fields by network devices and, consequently, the need to replace or upgrade network equipment.

Thus, one of the possible SDN evolution directions is separation of node-network and packet-switch interfaces in order to withstand the requirements of an ”ideal” network infrastructure. For this purpose the notion of core and edge of the network as well as with MPLS is introduced. In this case, all three interfaces are formalized and used each in its place. There by the devices ensure the functioning of the core and edge devices are controlled by a protocol similar to OpenFlow, but separately from each other, by means of separate software controller as the core and the boundary perform different tasks. Core of the network is responsible for transporting data (uni- and multicast) and for intellectual queuing policy in the event of an overload of certain directions.

The boundary of the network, and devices placed in it are responsible for the entire spectrum of network services demanded by modern applications — isolation, security, quality of service, availability of internal servers [14]. Therefore, an edge device is also managed separately from the core and uses core services to forward traffic.

Using sectioning in network management scheme is closely related to building of distributed controllers. Pursuing tasks of building robust controller researchers came to conclusion that network view sectioning for every invocation of controller along with means of intercontroller interaction is required. Good examples of works in this direction are Kandoo [15] and Onix [16].

IV. Event registration module EventBus

We have developed a scheme of interaction between the various modules implemented on SDN controller. This scheme is centered around linkage module registering events from other modules (Fig. 2). It will allow more flexibility in event handling and interactions between modules. Each module runs in daemon mode, that is runs in the background without direct user interaction. EventBus module is implemented on Node.js

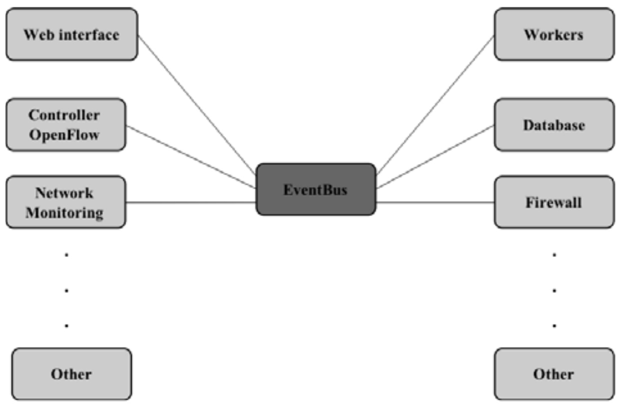

Fig. 2. Scheme of interaction between applications on the SDN controller

platform, this framework is very important as it provides improved performance in socket manipulation.

All modules implemented on SDN controller, are to be interconnected by EventBus module, allowing more flexibility in event handling and interactions between them. Each module connects and maintains the connection to EventBus module via local socket (unix domain socket).

The principle of EventBus operation (Fig. 3). Each module can send events, receive events, call the functions of another module using remote procedure call (RPC). Each event can also be accompanied by data. If a module wants to receive a certain type of events, it informs EventBus, which remembers the recipient for the specified event type. If a module wants to inform about an event, it sends a signal to EventBus, which in turn looks which modules are subscribed to the desired event type and forwards information to the set of recipients. If the module wants to call a function in another module, it also sends a signal to EventBus and waits for a response. EventBus at the same time invokes the corresponding function at its destination module, receives the response and forwards it to waiting module. Remotely callable functions must be registered in EventBus.

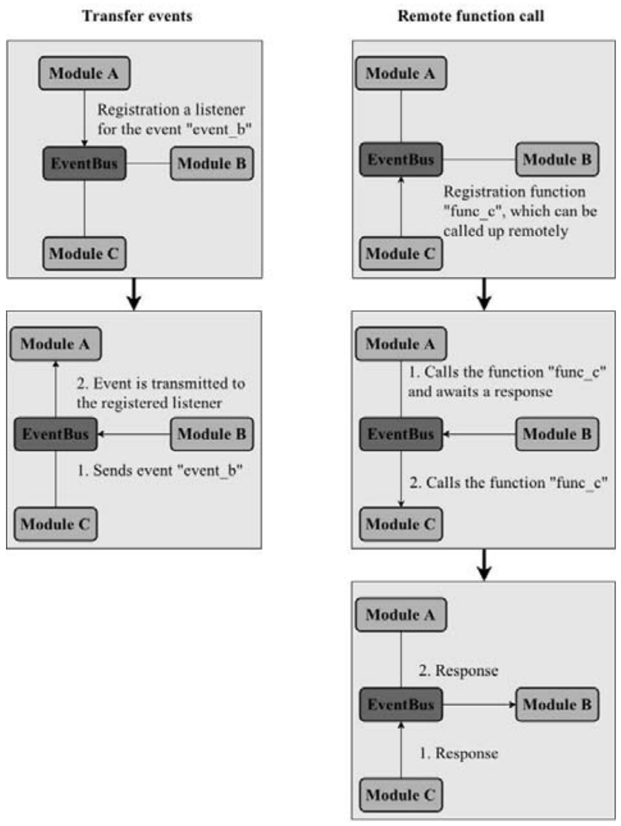
Transfer events
Remote function call
Module A
Registration a listener for the event "event_b"
EventBus
Module B
Module C
Registration function "func_c", which can be called up remotely
2. Event is transmitted to the registered listener
1. Sends event "event_b"
1. Calls the function "func_c" and awaits a response
2. Calls the function "func_c"
2. Response
1. Response

Fig. 3. Scheme of the module event registration

Since all modules are connected by means of EventBus module, it is better to include EventBus into web-based interface that allows users to manage the SDN controller. We have developed a web interface module, which is implemented in Python using the library gevent. Client-side executable, which runs in browser is implemented on with ExtJS 4.1 library. Web interface is used to display user interface in a browser. The user interface is needed to configure and manage SDN controller, to display statistics in a convenient form, visualize topology of SDN. The module has two parts: a web server and EventBus client. The web server is responsible for transferring the content to user browser and alerting the browser on new events. During the development we used out-of-the-box web server, which is part of gevent library.

To dynamically display data in a browser we built interaction scheme of the browser and the web server based on websocket protocol. We used Python library gevent-socketio, which provides event transfer functionality over websocket protocol (Fig. 4).

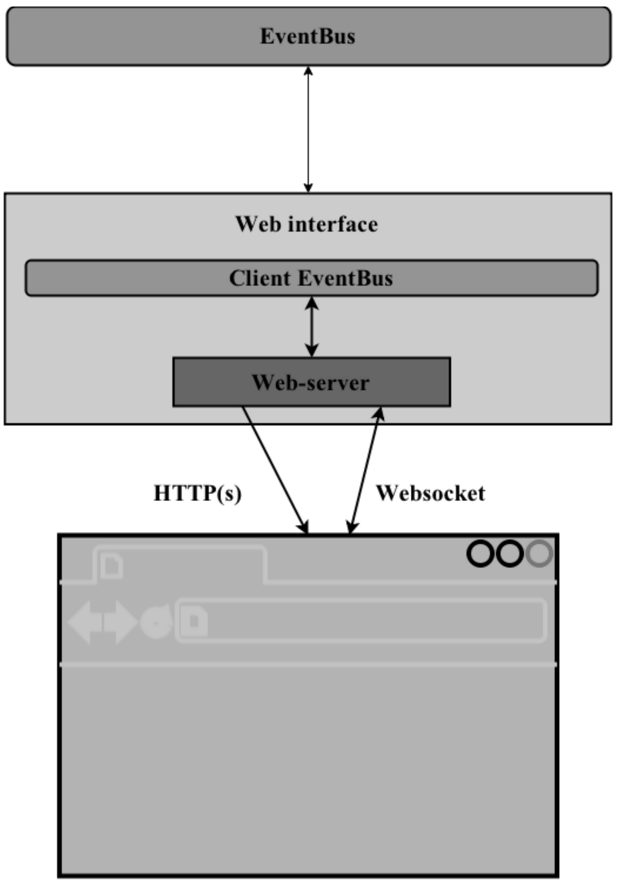

Fig. 4. The general scheme module operation EventBus

The user interface has the following menu items:

- Users. Here, SDN users are configured. When adding new user the following attributes are to be specified: a unique user name used when creating reports and statistics in the event log; IP-address; also there are optional descriptive fields.
- Network Configuration. This menu section is used to configure rules for OpenFlow switches (static routes, activated ports), specify the address of the local network. Local network address is configured to determine a network that will be serviced and contain users set in the above section. Setting the rules of an OpenFlow switch is only possible if the switch has been detected by the system. To make the settings you have to choose from list of available OpenFlow switches. When configuring the OpenFlow switch information on the active / inactive ports is immediately available. Each OpenFlow switch can be assigned a name and description (for example, physical location). The names of the switches are used in the event log and statistics for better readability. If the name is not configured, datapath_id (unique identifier) of the switch required by standard OpenFlow protocol standard is used.
- Firewall. In this menu section packet filtering on OpenFlow switches is configured. Filter rule may include the following parameters: incoming or outgoing port on the switch, MAC or IP address of source or destination address, TCP or UDP port of source or destination. Configuration of packet filtering rules on an OpenFlow switch is only possible if the switch has been detected by the system. To configure choose from the available OpenFlow switches list.
- Topology. This section of the menu provides the visual connection diagram of OpenFlow switches. When this menu section is opened web interface module requests the current network topology from OFC, processes this information and sends it to the user interface module, which depicts the graphical view of the network. Also this scheme reflects important events in the SDN, for example, loss of access to one of the switches in the SDN will be reflected in the scheme by red highlighting of this switch. With this menu section, you can configure the network, configure the firewall on any available OpenFlow switch and view statistics of any switch registered in the system. To do this, one needs to open the menu by right-clicking on the image of the desired switch, select the appropriate item. After that there will be a transition to the corresponding item in the main menu: Network Setup, Firewall, Statistics.
- Monitoring. This menu item displays graphs showing: server CPU utilization, consumption of both virtual and physical memory on the server, load level of the operating system. To view statistics of a particular OpenFlow switch choose it from the list of OpenFlow switches detected by the system.
- Statistics. In this menu section, you can get statistics on traffic consumed by users, on amount of traffic which has crossed a particular switch, the number of incoming / outgoing packets on each port of each OpenFlow switch. The statistical data can be sorted by time intervals of 5 minutes, 1 hour, 1 week, 1 month, 1 year. To view statistics of a particular OpenFlow switch choose it from the list of OpenFlow switches detected by the system. Similarly, you view statistics for users registered in the sysytem.
- Event log. In this menu section you can view the events that occurred in the SDN network (e.g., loss of access to a switch, connection of a new device to a switch port) or in the system itself (for example, lack of disk space, events generated by system modules). For convenience, there are two tabs, one shows events in the SDN, another in the system.

V. Conclusion

Thus, we can say that the module EventBus is the equivalent of a northbound SDN controller with a single web-interface. Currently, there are no specifications or other documents governing the creation and principles of northbound SDN controller. In this connection our future research will focus on the creation of northbound, based on the principles of work developed module EventBus, for SDN controller FloodLight [17]. The selection of the FloodLight is explained by the fact that it is free, undistributed and widely supported controller. Northbound, based on the principles of operation of the module EventBus provides any SDN controllers scalability and distribution, which is one of the fundamental problems in the field of software defined networks.

This work was performed in the Yaroslavl State University with support of the Ministry of Education and Science of the Russian Federation in the framework of the Subsidy Agreement (ID RFMEFI57414X0036).

References

[1] K. Hafner, M. Lyon, "Casting the Net", *The Sciences*, vol. 36, issue 5, Sep.-Oct. 1996, pp. 32-36.

[2] M. A. Nikitinskiy, D. Ju.Chalyy, "Performance analysis of trickles and TCP transport protocols under high-load network conditions", *Automatic Control and Computer Sciences*, vol. 47, issue 7, Dec. 2013, pp. 359-365.

[3] B. Furht and A. Escalante, *Handbook of cloud computing*. Springer, Sep. 2010, p. 634.

[4] V. Sokolov, I. Alekseev, M. Nikitinskiy, D. Mazilov, "A network analytics system in the SDN", *SDN&NFV: The Next Generation of Computational Infrastructure: 2014 International Science and Technology Conference Modern Networking Technologies (MoNeTec)*, Oct. 2014, pp. 160-162.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, number 2, Apr. 2008, pp. 69-74.

[6] P. Kazemian, G. Varghese, N. McKeown, "Header Space Analysis: Static Checking For Networks", *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, Apr. 2012, pp. 113-126.

[7] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford, "A NICE Way to Test OpenFlow Applications", *NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, Apr. 2012, pp. 127-140.

[8] A. Khurshid, W. Zhou, M. Caesar, P. B. Godfrey, " VeriFlow: verifying network-wide invariants in real time", *HotSDN '12 Proceedings of the first workshop on Hot topics in software defined networks*, Aug. 2012, pp. 49-54.

[9] OpenFlow Switch Specification, Version 1.3.4, Web: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.4.pdf.

[10] OpenFlow Switch Specification, Version 1.4.0, Web: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[11] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A. W. Moore, *PAM'12 Proceedings of the 13th international conference on Passive and Active Measurement* , Mar. 2012, pp. 85-95.

[12] P. Bosshart, G. Gibb, Hun-Seok Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, M. Horowitz, "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN", *ACM SIGCOMM Computer Communication Review*, vol. 43, issue 4, Oct. 2013, pp. 99-110.

[13] M. Casado, T. Koponen, S. Shenker, A. Tootoonchian, "Fabric: a retrospective on evolving SDN", *HotSDN '12 Proceedings of the first workshop on Hot topics in software defined networks*, Aug. 2012, pp. 85-90.

[14] M. Nikitinskiy, I. Alekseev, "A stateless transport protocol in software defined networks", *SDN&NFV: The Next Generation of Computational Infrastructure: 2014 International Science and Technology Conference Modern Networking Technologies (MoNeTec)*, Oct. 2014, pp. 108-113.

[15] S. Yeganeh, Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications", *HotSDN '12 Proceedings of the first workshop on Hot topics in software defined networks*, Aug. 2012, pp. 19-24.

[16] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H.Inoue, T. Hama, S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks", *OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation*, Oct. 2010, pp. 351-364.

[17] Floodlight SDN OpenFlow Controller, Web: https://github.com/floodlight/floodlight