Telfor Journal, Vol. 10, No. 1, 2018.

Two-Phase Load Balancing for Data Center Networks using OpenFlow

Nataša Maksić, *Member, IEEE*

***Abstract* - This invited paper describes an OpenFlow solution for two-phase load balancing optimized for data center networks. The solution uses aggregate flows and Proxy ARP procedure in order to achieve reduced size of OpenFlow tables, fast flow configuration and compatibility with standard Ethernet networks. The paper presents modification of the proposed solution, based on configuration of interconnected groups of aggregate flows. The paper also discusses the scalability of the solution.**

***Keywords* — OpenFlow, data center, load balancing.**

I. Introduction

OPENFLOW protocol was introduced by academic community which needed the way of controlling packet switches for research purposes [1].

Traditionally, packet switches are closed devices with proprietary hardware and software. Hence, researchers cannot modify the behavior of packet switches. With the introduction of OpenFlow they gained a tool with which they could create and control packet flows in the network. OpenFlow is standardized by Open Networking Foundation [2].

The OpenFlow standard consists of two main parts: specification of switch functionality and specification of OpenFlow protocol.

In terms of switch functionality OpenFlow standard defines flow tables, group tables and meter tables. Flow tables enable matching packets to flows, and performing actions such as packet modification and packet forwarding. Flow tables also contain counters which enable insight into network utilization.

In group tables, each entry contains sets of actions. These sets are referred to as buckets. Instead of defining actions, flow table entries can refer to group table entries. Buckets enable operations such as broadcast, multicast, packet based load balancing and fast failover.

Meter tables enable flow rate limiting. A flow table entry can reference a meter table entry, and a meter table entry limits the aggregate throughput of all flows which reference it.

Paper received April 1, 2018; revised June 22, 2018; accepted June 28, 2018. Date of publication July 31, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Grozdan Petrović.

This paper is a revised and expanded version of the paper presented at the 25th Telecommunications Forum TELFOR 2017 [14].

This work was supported under the project TR-32022 by the Serbian Ministry of Science and Education, and by companies Telekom Srbija and Informatika.

Nataša D. Maksić is with the School of Electrical Engineering, University of Belgrade, Bul. kralja Aleksandra 73, 11120 Belgrade, Serbia (e-mail: maksicn@etf.rs).

The second major component of OpenFlow standard is OpenFlow protocol. With OpenFlow protocol we can establish communication with switch, modify OpenFlow tables in switch in order to configure flows, and collect flow statistics.

In OpenFlow, algorithm for creating packet flows is implemented in a controller. A controller is an application which communicates with OpenFlow switches using OpenFlow protocol. A controller is typically executed on a general purpose computer.

Open-source extendable controller applications written in different programming languages have emerged in previous years. These applications handle low-level details of OpenFlow protocol communication and provide higher level APIs for configuration of OpenFlow tables in switches. For the implementation of two-phase load balancing we have used Ryu controller [3]. Ryu is supported by telecommunications company NTT. It is written in Python programming language.

This paper presents an OpenFlow solution for two-phase load balancing. The two-phase load balancing improves the distribution of packet traffic across the network by using an intermediate routing point [4, 5, 6]. Implementation of two-phase load balancing based on loose source routing is presented in [7].

One important advantage of using OpenFlow is the possibility to use lower cost switches that do not implement complicated protocols. Another advantage is the possibility to modify the routing algorithm only by modifying the controller. Such modification is not possible with switches that have embedded software which is not accessible to users.

Paper [8] presents an OpenFlow routing solution for fat-tree datacenter network topologies [9]. This solution performs load balancing by measuring link utilizations using OpenFlow. Load Balancing is performed by using Dynamic Load Balancing (DLB) algorithm proposed in that paper. Another OpenFlow load balancing proposal specialized for fat-tree networks with different path selection algorithm, Global Load Balancing (GLB), is presented in [10]. In comparison with our proposal, proposals [8], [10] are limited to fat-tree topology, and they do not use aggregate flows or Proxy ARP procedure.

OpenFlow solution for Multipath TCP path assignment based on network topology is presented in [11]. This solution selects paths by minimizing the number of links shared between paths and path length. Multipath TCP proposal [12] uses the existence of multiple paths in data center networks and achieves more even network utilization by creating TCP subpaths.

Hedera [13] uses OpenFlow in order to monitor flows and to transfer them to an appropriate path once they reach certain throughput.

This invited paper is an extension of the paper published at Telfor conference [14]. This extended version includes the explanation of throughput measurement for flows and links, explanation of flow path selection algorithm and the criteria for triggering recalculation of routing parameters. It also explains the packet forwarding with multiple aggregate flows in hierarchical data center topologies and discusses the benefits of integration of application with the controller. The overview of solution scalability is also included.

Section II of this paper describes possible paths of the packet through the network when using two-phase load balancing. Section III describes OpenFlow solution for packet forwarding based on two-phase load balancing. Section IV discusses assignment of packet flows to paths using OpenFlow. Section V describes the procedure for discovering server location in the network. Section VI discusses the measurement of flow throughputs and link throughputs using OpenFlow. Section VII discusses selection of the balancing router for the new flow. Section VIII discusses the criteria for recalculation of LB-ECR routing parameters. Section IX discusses the reduction of OpenFlow table sizes by creating groups of aggregate flows. Section X discusses the possibility of integrating communicating applications with the controller. Section XI discusses the scalability of the proposed solution. Section XII contains the conclusion.

II. Two-phase Load Balancing in Data Center Networks

Data center networks contain a large number of servers connected by a communication network. In this paper we describe OpenFlow load balancing solution for data center networks.

Various topologies have been proposed for data center networks [9], [15], [16]. These topologies use cost-effective high-radix switches in order to provide enough bandwidth for server communication. The resulting communication network has multiple communication paths between server pairs. In order to utilize network evenly, load balancing is needed.

Fig. 1 shows fat-tree topology with three levels and twenty switches. Switches are represented with circles.

In fat-tree topology, servers are connected only to switches in the bottom layer. In Fig. 1, applications executing on servers are represented with triangles. These applications generate application flows, which are transported through the fat-tree topology.

OpenFlow controller is shown in the upper part of Fig. 1. It communicates with switches in order to monitor and configure traffic processing and forwarding.

ECMP is one solution for improved load balancing. It performs balancing along paths with minimal cost. This leaves the possibility of improvement which would perform load balancing over non-minimal cost paths. One such improvement is LB-ECR [6], which combines ECMP and optimized two-phase load balancing in order to enable more even utilization of data center network and prevent congestion.

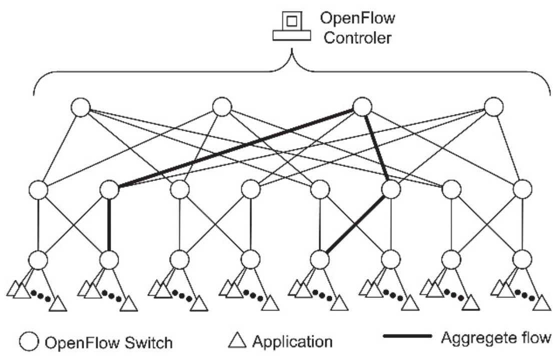

Fig. 1. Fat-tree topology with OpenFlow controller.

In LB-ECR a packet is routed from the source to the balancing switch and then to the destination. ECMP is used on both parts of the path. Optimization procedure is used to calculate which part of traffic will be routed over which balancing switch.

LB-ECR uses a linear programming optimization procedure in order to calculate how much traffic should be directed to which balancing switch. The part of traffic that should be directed to a balancing switch is called a balancing coefficient of that balancing switch. In LB-ECR, each switch has one balancing coefficient, i.e. every switch directs the same amount of traffic to this balancing switch. An alternative approach is to have one set of balancing coefficients for each pair of communicating switches [4]. LB-ECR uses one set of balancing coefficients due to the symmetry of the data center topologies, and the need to reduce the size of the optimization model and calculation time of balancing coefficients.

In this paper we propose an OpenFlow solution for flow-based load balancing. In flow-based load balancing packets belonging to one application flow are routed together over the same path. In order to accomplish load balancing different application flows need to be assigned to different paths. The advantage of flow-based load balancing is in-order delivery of packets. In addition, flow congestion control is suited for transmission over a single path.

We configure one OpenFlow path for each possible path of the packet. Since this path will carry traffic of multiple application flows, we will use the term aggregate flow to denote the traffic forwarded along the OpenFlow path. Each aggregate flow will incorporate one of the ECMP paths from the source switch to the balancing switch and one of the ECMP paths from the balancing switch to the destination switch.

Fig. 1 illustrates one aggregate flow. This flow starts at one of the bottom switches, and continues to the balancing switch located in the upper layer. Then it continues towards another bottom switch. The links belonging to this flow are shown in bold.

Aggregate flow shown in Fig. 1 corresponds to one of the balancing paths between two switches in the lower layer. This aggregate flow is configured as one OpenFlow flow that carries all packets from source switch to destination switch assigned to this balancing path. Such aggregate flow eliminates the need for configuration of all OpenFlow tables for each new application flow assigned to an observed balancing path.

In the next section, we will discuss the OpenFlow configuration of aggregate flows from the source switches over the balancing switches to the destination switches.

III. Aggregate flow configuration

In order to configure aggregate flows between switches we extend the packet header with the information needed for routing based on flows. The OpenFlow standard supports multiple packet header extensions such as VLAN tags and MPLS and PBB headers. In this implementation we use VLAN tags.

Since aggregate flows are known in advance, they are configured proactively, after the calculation of balancing parameters. By configuring aggregate flows, addition of new flow requires configuration only in switches attached to communicating servers. The introduction of aggregate flows will reduce the size of OpenFlow tables and the time needed to configure a new application flow.

The OpenFlow standard supports actions for adding and removing VLAN tag. Switches use one value of VLAN tag for each aggregate flow between source switch over balancing switch to the destination switch. A source switch adds VLAN tag which will be used for forwarding a packet towards the balancing switch. Based on this tag, a packet will be forwarded through the network to the destination switch. Finally, the destination switch removes VLAN tag and forwards the packet to the port on which server is attached.

Ryu controller provides API which enables us to configure forwarding based on VLAN tags. The Python code that creates a flow table entry for packet forwarding based on VLAN tag is shown in Fig. 2.

```
match = ofparser.OFPMatch()
match.set_in_port(in_port)
match.set_vlan_vid(
  vlan|ofproto_v1_3.OFPVID_PRESENT)
actions = [ofparser.OFPActionOutput(out_port)]
inst =[ofparser.OFPInstructionActions(
  ofproto.OFPIT_APPLY_ACTIONS, actions)]
mod = ofparser.OFPFlowMod(datapath=datapath,
  priority=1, match=match, instructions=inst,
  idle_timeout=0)
datapath.send_msg(mod)
```

Fig. 2. Creation of flow table entry for packet forwarding based on VLAN tag.

Object *match* is configured with input port of the packet (*in_port*) and the VLAN tag of the packet (*vlan*). The flow table entry will be executed for packets arriving at port *in_port* with the VLAN tag *vlan*.

Variable *ofparser* is a reference to a module that contains implementation of OpenFlow version which is used. We have used OpenFlow 1.3.

List *actions* is initialized with action *OFPActionOutput* and output port to which a packet will be forwarded *out_port*. Object *inst* contains an instruction to execute action in case of match.

Representation of OpenFlow flow table entry is created in object *mod*. Object *datapath* contains the identifier of OpenFlow instance, which is a 64-bit number. Besides match and actions, we define priority of the table entry as 1 and idle timeout as 0. The value of 0 for *idle_timeout* defines that a flow entry will not be removed in case of flow inactivity. A value greater than zero defines the number of seconds after which the flow will be deleted if no packets arrive. Finally,an OpenFlow command for the table configuration is sent. An example of flow table entry created using previous commands in the Open vSwitch is: *cookie = 0x0, duration = 2267.977s, table = 0, n_packets = 91965, n_bytes = 6437558, priority = 1, in_port = 4, dl_vlan = 2065 actions = output:1.*

With the definition of the VLAN tags we have a means of matching packets in order to forward them. In the next section, we will specify criteria for grouping packets to application flows and assigning application flows into aggregate flows.

IV. Application Flow configuration

An application flow needs to be assigned to an aggregate flow in the switch connected to the server. In the proposed solution all servers are located in one Ethernet LAN.

When a new flow is initiated between two applications executing on servers, a MAC address of the destination server may not be present in ARP cache and an ARP request is generated.

When the switch receives an ARP request, it will forward this request to the controller. In our implementation these requests are automatically forwarded to the controller since they are not assigned to any flow. The controller will inspect the message, and check whether the destination IP address belongs to a known server. The controller keeps Ethernet addresses and IP addresses for all servers which have communicated previously.

If the controller has knowledge of the destination server then it will reply to the ARP request with the ARP response packet containing the MAC address of the switch port to which server is connected. This will result in a server sending the packet to the switch and prevent ARP request from entering the network.

After the server has acquired the MAC address of the switch it will start sending packets which belong to flow and have destination IP addresses of the other server. These packets may be UDP or TCP packets or ICMP packets like ping. With the information from this first packet, the controller can create flow table entries both in the switch connected to the source server and in the switch connected to the destination server.

For UDP, TCP and ICMP packets, match rules for packets arriving from the server contain an input port, EtherType field of Ethernet header, source and destination IP address and IP protocol type. In the case of UDP and TCP packets it also contains source and destination UDP/TCP port number.

For packets arriving from the network and destined to server, match rules include a VLAN tag and also contain the rules for matching EtherType field of Ethernet header to IPv4, source and destination IP address and IP protocol type. For UDP and TCP packets, match rules include source and destination UDP/TCP port number.

Action list for creating a flow contains several actions. For the packets that are received from the server the first rule is to replace the destination MAC address with the

MAC address of the destination server. This is needed because the server will send a packet to the MAC address of the switch. Then, the VLAN tag is added to the packet and the packet is sent to the outgoing port.

Code sections for the configuration of application flows in Ryu controller are shown in [14].

The case when the destination server is unknown to the controller is described in the next section.

V. Server location

If the destination server of ARP request is not known to the controller, then the controller will put the ARP request in the waiting list and perform search for the destination server. This search is performed by generating ARP requests in the controller for all the switches and instructing the switches to send them to ports to which servers are connected.

After the arrival of ARP response from any of the servers to its connected switch, the switch forwards the response to the controller, and the controller will add the server to the list of known servers. Then, the controller will search the ARP request waiting list, and send ARP response to the appropriate server. This response contains MAC address of the switch and it will enable the server to start sending packets belonging to the flow and the controller will continue with the procedure described in the previous section.

Extracting data from incoming ARP packets and generating new ARP packets is implemented by using Ryu libraries. Ryu libraries enable data extraction and packet generation for various protocols. The example shown in Fig. 3 illustrates the creation of ARP packet.

The first line in Fig. 3 creates an object for the Ethernet packet header with given source and destination MAC addresses and with EtherType field of Ethernet header value set to ARP protocol. The second line creates an object for ARP packet with the given parameters. By using values 1 or 2 for *opcode* parameter we can create ARP request and response messages. The following lines in Fig. 3 create and encode ARP packet.

VI. Measurements of throughputs

Ryu controller provides API which enables periodic reading of transmitted packets and transmitted bytes per flow and per port. Throughputs of application flows and links are calculated by reading corresponding byte counters. Ryu function *OFPFlowStatsRequest* is used to read values of byte counters assigned to a flow. Ryu function *OFPPortStatsRequest* is used to read values of byte counters assigned to a port.

```
e = ethernet(dstMac, srcMac, ether.ETH_TYPE_ARP)
a = arp_ip(opcode, srcMac, srcIp, targetMac, targetIp)
p = Packet()
p.add_protocol(e)
p.add_protocol(a)
p.serialize()
```

Fig. 3. Creation of ARP packet.

Based on two consecutive values of byte counters, a controller can calculate average throughput for an application flow or link. Throughput is calculated by subtracting two consecutive values of byte counters by the time interval between the arrivals of OpenFlow messages containing these counters. This principle for calculating throughputs is also used in SNMP, which provides byte and packet counters [17].

Byte counters assigned to flows are used for measurement of flow throughputs. These measurements are needed for the selection of flow paths described in section VII.

Byte counters assigned to switch ports are used for the measurement of link throughputs. These measurements indicate the need for the recalculation of LB-ECR balancing coefficients, as described in section VIII.

VII. Selection of flow path

Each of the configured aggregate flows carries packets over one of the balancing switches. Each assignment of application flows to aggregate flows needs to be such that a portion of the source switch traffic directed towards balancing switch corresponds to a calculated optimal value. In order to achieve that, the controller measures throughput of each flow and calculates throughput towards each of the balancing switches.

When a new flow is accepted in a switch it is directed towards a balancing switch which has the largest difference between throughputs calculated by LB-ECR and measured throughputs. The process of assigning an application flow to an aggregate flow is described in section IV.

Different flows may reach different levels of throughput. This depends on the communicating applications. For this reason, it may be necessary to transfer application flows from one balancing switch to the other.

The operation of changing a balancing switch to which a flow is directed is performed only in the switch through which a flow enters the network. This is performed by changing the aggregate flow to which the application flow is assigned. Since a destination aggregate flow is already established, packets will immediately be transferred to a new path. This results in a fast reconfiguration with no packet loss, and with only one OpenFlow configuration message.

Finally, the recalculation of balancing coefficients may lead to a difference between throughputs calculated by LB-ECR and measured throughputs. In this case a controller will redirect some application flows from one balancing router to another, i.e. from one aggregate flow to another.

VIII. Recalculation of LB-ECR balancing coefficients

Inputs to LB-ECR optimization algorithm are the traffic load that each switch inserts from external hosts into the load-balancing network, the traffic load that each switch receives from the load-balancing network and forwards to external hosts, as well as paths of the underlying routing protocol. When the relative amounts of inserted and terminated traffic loads on different switches change, then the optimal values of balancing coefficients will also change and balancing coefficients might need to be recalculated in order to avoid congestion.

Routing traffic with slightly inaccurate values of balancing coefficients may not lead to congestion. In order to prevent unnecessary calculation of balancing coefficients, and consequent redirections of flows, the controller continuously measures link utilizations.

LB-ECR calculations provide the relative values of link utilizations in the network. When the distribution of application traffic changes, the relative link utilizations will also change, and the controller will trigger the recalculation of balancing coefficients when this change reaches a predefined threshold.

IX. Hierarchical data center topologies

Data center topologies are intended to provide high throughput to all applications executing on connected servers. With this property, applications can be executed on any server in data center, and transferred from one server to another without the need to take topology into account. This requirement results in highly symmetrical data center topologies.

Data centers topologies may have a large number of servers and switches. For data centers with a large number of switches, authors have proposed hierarchical topologies such as Dragonfly [16]. In such topologies, the number of aggregate flows may be reduced by introducing higher level aggregate flows, which are established on the links belonging to a higher hierarchical level.

This can be accomplished by the controller since it knows the entire path of the application flow and can assign the application flow to aggregate paths of different levels along the path from source to destination switch.

We will illustrate this principle on the example of Dragonfly topology [16]. Dragonfly topology is divided into groups. Groups have intragroup topology. Every group is connected to other groups with intergroup topology. One example of these topologies is fully connected intergroup topology and flattened butterfly [15] intragroup topology. Fig. 4 illustrates Dragonfly topology.

Since group topologies are well defined, we can generate aggregate flows for all paths within each of the group topologies. Additionally, we can create aggregate flows in intergroup topology, which would carry application flows between group topologies.

In this configuration the controller needs to create the transfer of application flow between aggregate flows. An example for this operation in Dragonfly topology would be the transfer from link within intragroup topology to link within intergroup topology.

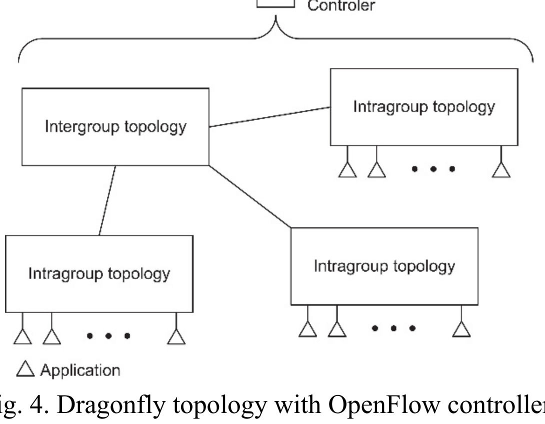

Fig. 4. Dragonfly topology with OpenFlow controller.

Match criteria for transferring an application flow from one aggregate flow to another is shown in Fig. 5. Controller uses VLAN tag of incoming aggregate flow and values of the IP and, in this case, UDP headers that define application flow.

```
match.set_in_port(in_port)
match.set_dl_type(ether.ETH_TYPE_IP)
match.set_ip_proto(inet.IPPROTO_UDP)
match.set_ipv4_src(self.ipv4_to_int(pkt_ipv4.src))
match.set_ipv4_dst(self.ipv4_to_int(pkt_ipv4.dst))
match.set_udp_src(pkt_udp.src_port)
match.set_udp_dst(pkt_udp.dst_port)
match.set_vlan_vid(vlanid|ofproto_v1_3.OFPVID_PRESENT)
```

Fig. 5. Match rules for creation of TCP flow.

Actions for transferring an application flow from one aggregate flow to another are shown in Fig. 6. In these actions VLAN tag of the previous aggregate flow is replaced by VLAN tag of the next aggregate flow. Finally, packet is forwarded to the output port according to path of the current aggregate flow.

```
actions = [
    ofparser.OFPActionSetField(vlan_vid=(vlanid
     | swDatapath.ofproto.OFPVID_PRESENT)),
    ofparser.OFPActionOutput(out_portsecond, 0)]
```

Fig. 6. Action list for packets changing aggregate flow.

Introduction of paths which encompass multiple aggregate flows also changes the procedure for transferring an application flow from one balancing router to another. For this operation, the controller needs to set all connections between aggregate flows. In order to avoid traffic loss, the controller firstly needs to set connections on the new path, without changing the settings in the first switch of the application flow path. Then the controller can transfer the flow to the new path by changing settings in the first switch as described in section IV. Finally, the controller can remove connections between aggregate flows on the old path.

The possibility of composing application flow path from multiple aggregate flows introduces the tradeoff between the total number of aggregate flows and the number of configuration actions. In large data centers it may be important to reduce the total number of aggregate flows by introduction of multiple groups of aggregate flows. These domains are intrinsic to many data center topologies in order to save resources. One of the motivations behind Dragonfly topology was to save cable length by dividing topology into groups.

X. Integration of applications and controller

In general, the number of application flows communicating through data center topology may be large. With aggregate flows, the application flows only need to be configured in switches through which they enter or leave the topology. However, the number of flows entering or leaving the topology in a switch may be large, and the OpenFlow controller needs to monitor the throughputs of these flows in order to provide correct flow path selection, in accordance with OpenFlow balancing coefficients.

The number of byte counters that a controller needs to read from a switch may be reduced by introducing communication between applications and controller. If an application would inform the controller about the maximal throughput of the new flow, the controller would not need to measure the throughput of this flow. This is possible in a controlled and closed data center environment, with applications specialized for execution in data centers.

XI. Scalability analysis

As far as the size of flow table is concerned, each switch stores one entry for each aggregate flow that traverses that switch. As explained, aggregate flows connect end-switches over balancing switches. The number of aggregate flows traversing each particular switch depends on the topology. However, with aggregate flows, the size of flow tables does not depend on the actual number of application flows in data center. The calculation of the number of flows needs has to be performed for each topology and each set of balancing coefficients. For example, in the fat-tree topology shown in Fig. 1, for the same traffic demands in each bottom switch, only top level switches have balancing coefficients different from zero. In this case each switch in the bottom level has two unidirectional connections over each top level switch with each bottom level switch, resulting in 56 aggregate flows per bottom level switch. Each middle level switch also has 56 unidirectional aggregate flows and each switch in top level has 112 unidirectional aggregate flows.

Flow tables in end-switches also need entries for mapping application flows to aggregate flows. However, the number of these flows is not directly dependent on the topology size, but only on the functionality of applications executing on the servers connected to the switch.

When aggregate flow groups are used, the number of flow table entries in switches, in which application flows change aggregate flow, will depend on the number of application flows traversing those switches.

XII. Conclusion

OpenFlow enables complete control of the network traffic from the controller. In this paper we present OpenFlow solution for flow based two-phase load balancing. This solution is intended for data center networks. The proposed solution requires no modifications of servers, which communicate in one Ethernet network. By using LB-ECR, the proposed solution enables more even network utilization and congestion avoidance.

By using OpenFlow, switches in network can be less expensive because they do not have to implement complicated protocols. They only need to support OpenFlow and to be fast enough to support packet throughputs required in a data center. An additional advantage of using OpenFlow is the possibility of changing network operation by changing software of the controller, without the need for modification of switches.

Proposed solution reduces the size of OpenFlow tables and the number of configuration operations by introduction of aggregate flows. Additionally, it offers a tradeoff between the number of aggregate flows and configuration operations by using groups of aggregate flows.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan et al., “Openflow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] The OpenFlow Switch Specification, Available at: https://www.opennetworking.org/software-defined-standards/specifications/

[3] Ryu Project Team, “RYU SDN Framework”, Release 1.0, Available at: https://osrg.github.io/ryu-book/en/Ryubook.pdf

[4] M. Kodialam, T. V. Lakshman and S. Sengupta, “Maximum Throughput Routing of Traffic in the Hose Model,” *INFOCOM 2006*, Barcelona, Spain, April 23-29, 2006.

[5] M. Antić, A. Smiljanić, “Oblivious Routing Scheme Using Load Balancing Over Shortest Paths,” *Proceedings of the ICCC ‘08*, Bejing, China, May 19-23, 2008.

[6] N. Maksić, A. Smiljanić, “Improving Utilization of Data Center Networks,” *IEEE Communications Magazine*, November 2013, pp. 32-38.

[7] M. Antić, N. Maksić, P. Knežević, A. Smiljanić, “Two Phase Load Balanced Routing using OSPF,” *IEEE Journal on Selected Areas in Communications (J-SAC)*, January 2010, pp. 51-59.

[8] Y. Li, D. Pan, “ OpenFlow based Load Balancing for Fat-Tree Networks with Multipath Support,“ *IEEE ICC 2011*, Budapest, Hungary, 9-13 June, 2011.

[9] M. Al-Fares, A. Loukissas, A. Vahdat, “A Scalable, Commodity Data Center Network Architecture,” *SIGCOMM’08,* Seattle, Washington, USA, August 17–22, 2008.

[10] Y. Wu, W. Zhang, “OpenFlow-Based Global Load Balancing in Fat-Tree Networks“, *Advanced Materials Research* (Volumes 989-994), pp. 4794-4798, July 2014.

[11] C. Nakasan, K. Ichikawa, H. Iida, P. Uthayopas, “ A Simple Multipath OpenFlow Controller using topology-based algorithm for Multipath TCP“, *Concurency and Computation, Practice and Experience*, Volume 29, Issue 13, 10 July 2017.

[12] C. Raiciu et al. “Improving datacenter performance and robustness with multipath TCP,” *Proceedings of the ACM SIGCOMM*, Toronto, Canada, August 2011.

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang and A. Vahdat, “Hedera: Dynamic flow scheduling for data center networks,” *Proceedings of Usenix NSDI* 2010.

[14] N. Maksic, “OpenFlow-Based Load Balancing in Data Center Networks,” *Proceedings of Telfor 2017.*, Belgrade, Serbia, November 2017.

[15] J. Kim, W. J. Dally, D. Abts, “Flattened Butterfly: A Cost-Efficient Topology for High-Radix Networks,” *ISCA’07*, San Diego, California, USA, June 9–13, 2007.

[16] J. Kim, W. J. Dally, S. Scott, D. Abts, “Technology-Driven, Highly-Scalable Dragonfly Topology,” *ISCA '08*, 35th International Symposium on Computer Architecture, Beijing, June 21–25, 2008.

[17] K. McCloghrie and M. Rose, “Management Information Base for Network Management of TCP/IP-based internets: MIB-II”, RFC 1213, Internet Engineering Task Force, 1991.