Dynamic Link Aggregation in Software Defined Networking

Ronaldo Resende Rocha Junior, Marcos A. M. Vieira, Antonio A. F. Loureiro
Computer Science Department
Universidade Federal de Minas Gerais
Belo Horizonte, MG, Brazil
Email: {ronaldorrjr,mmvieira,loureiro}@dcc.ufmg.br

***Abstract*—The link aggregation technique not only provides robustness to computer networks, but can also be a solution to the link saturation problem. This technique combines several physical interfaces to create a virtual link, adding up their existing bandwidth. In addition to increasing the throughput for data transmission, such a technique provides a quick and transparent recovery if a particular link becomes unavailable. Considering that the use of Software-Defined Networking (SDN) in business environments increases every day, this research presents a way to create link aggregations in such environments. This brings a better availability of services, among other benefits. To do so, an architecture that allows automatic, scalable and self-adaptive link aggregations was defined and implemented. As a way of evaluating such implementation, three algorithms were created using different premises: Hash Table, Traffic Analysis and Virtual Round-Robin. All implementations were tested in virtual and real environments. In both, the Open vSwitch open platform was used for packet switching and the Ryu controller was chosen to control the switches.**

***Index Terms*—Communication systems, Computer networks, LACP, Link Aggregation, Software Defined Networking**

I. Introduction

The computer networks management is a big challenge because they have become complex systems [1]. For each network equipment installed on a data-center, administrators need to perform unique configurations to ensure their full operation [2].

The traffic bottleneck is one of the main existing problems in classic network environments [3]. This happens, for instance, when multiple users try to use a single server on a network, overloading the link that connects the switch to that server. It is important to optimize features, including bandwidth usage in data-centers environments. In addition, it is necessary to create environments that have dynamic, highly configurable and autonomously manageable topologies.

The link aggregation allows for a more fault-tolerant, higher-availability network, which is ideal in enterprise environments [4]. In classical switches, this technique is performed using the open protocol LACP (Link Aggregation Control Protocol) or proprietary protocols. However, the LACP has several limitations. The configuration is done manually. The maximum number of aggregated physical links is limited to eight. In addition, all network interfaces must operate at the same speed to be aggregated.

To solve all these problems, a system capable to perform link aggregation in SDN environments is proposed. Its implementation is done directly on the SDN controller. Three algorithms for aggregation are proposed and evaluated: Hash Table, which calculates a hash based on the parameters of the package (physical addresses and IPs); Traffic Analysis; and Virtual Round-Robin. In this way, aggregation can be performed dynamically and without the need for manual configuration by a network manager.

The contributions of this work are the following: (i) a system for using link aggregation inter-flow in SDN environments; (ii) proposal of three algorithms for link aggregation; (iii) evaluation of the three algorithms in virtual and real environments, including a comparison with LACP and the native usage of Linux Bounding.

The proposed system has several advantages over the state of the art: (i) it is possible to aggregate multiple groups on the same switch, (ii) the solution is protocol free, meaning that there is neither dependence on LLDP nor commercial protocols, (iii) better bandwidth distribution, (iv) the aggregation is autonomous and does not require the intervention of a network administrator, (v) open source, (vi) it is possible to have more than eight aggregated interfaces, (vii) the aggregated links may have different transmission speeds and still be aggregated.

The remainder of this article is organized as follows. First, Section II discusses related work. Then, Section III reveals the proposed architecture and the link aggregation algorithms. Section IV describes the experiments and the results obtained. Finally, Section V presents the conclusion and future work.

II. Related Work

The LACP [5] is an open protocol that allows the configuration of link aggregation in classic switches. Its latest version is the 802.1AX-2014 standard. This protocol calculates a hash from multiple source and destination parameters. The protocol chooses the local switch port for the flow from that hash, that is, all the packets of this flow are forwarded only in that interface. When a new flow comes up, a new hash is calculated and probably another interface will be used for it. Thus, LACP uses all the aggregated interfaces, spreading the flows between them. The LACP protocol has some limitations: the network administrator must perform the configuration manually; the amount of physical interfaces can not be greater than eight;

978-1-5386-6950-1/18/$31.00 ©2018 IEEE

all switch interfaces must be configured at the same speed. The limitation of the number of interfaces can be overcome by a custom implementation (made in Juniper equipment, for example). But this approach brings interoperability issues, creating the need to use only equipments from the same manufacturer.

In [6], the latest searches in the SDN networks area are discussed and exposed. The authors [7] also highlight the benefits and differences between the classic model and the SDN. In the mini-course [8], the usage of tools like Mininet and POX is explored and exemplified. The work carried out by [3], although relatively old, elucidated the real problems of the bottleneck in computer networks.

The Open vSwitch platform was designed to be a virtual switch capable of running in any environment [9]. Since it was quite complete and robust, it was soon incorporated into SDN networks. However, when considering the link aggregation technique, this platform is not effective. The maximum amount of aggregated physical ports is limited, and cannot exceed four units (a limitation of the platform itself). In addition, this setting needs to be manual.

In the Linux operating system, there is a native library for link aggregation [10]. The link aggregation behavior depends on the method selected, but the main ones are known as balanced TCP and balanced SLB. The first uses packet information from layers two, three and four to balance traffic and depends on the other end (usually a switch), which needs to have the LACP enabled and configured. The latter uses only layer two information and is not dependent on the LACP protocol.

The authors in [11], [12], [13] and [14] consider the path of a TCP traffic between multiple links, without considering a possible local aggregation between them. However, they all consider paths through multiple hops. The difference between this work and the others is the approach in the layer two, that is, the communication between two switches with links aggregated between them.

The work carried out by [15] proposes the implementation of a platform capable of creating and managing virtual SDNs. Both the topology and the addressing scheme of such virtual networks are completely independent of each other. We believe that our work has applicability in environments that use OVX, since the physical topology necessary to support the virtual topology could benefit from the link aggregation technique.

In order to perform the package transmission using several paths, the authors in [16] propose the implementation of an improved version of the TCP protocol in data-centers environments. It is a similar approach to this work, but uses a different technique than link aggregation. In the same way, [17] presents a platform called FlexForward which is also directed to data-centers environments. However, its implementation is focused on an Open vSwitch alteration.

III. Inter-Flow Implementation

The system adopts the same SDN architecture where there is a controller that sends rules to the switch tables. It was developed using the Ryu controller, which manages all the details of connections between switches, interpreting and converting network packets into easy-to-use objects.

The developed application listens for incoming packet messages in the switch, maintains an internal table of MAC addresses of all connected stations, and adds the packet switching rules in the switches as connections are identified. For this, the OpenFlow protocol is used.

Basically, the application performs the following steps: (i) the application registration and initialization happen before the switches join the Ryu controller domain and allow the application instance to initialize the data that will be shared across the network. For example, the switch application initializes a table to keep information of the hosts MAC addresses and their respective ports. Another table is used to keep the local switch ports information. (ii) the initialization of a switch that connects to the Ryu controller occurs when a switch joins in the Ryu controller domain and the application checks all its characteristics through the event *EventOFPSwitchFeatures*. Also, in this process, the initial packet switching rule is added in all switches, forcing them to send the first set of packets to be analyzed by the controller. (iii) During the identifying link aggregations, a data structure is created to configure and maintain all link aggregations across the controllers domain. (iv) Finally, during the packet sending, the application monitors the event *EventOFPPacketIn*, which is responsible for analyzing unknown sent packets. This always happens when the switch does not have a specific rule to send that packet or if it has a general forwarding rule that instructs it to send the packet to the controller to parse it.

A. Main events processed by the controller

All events generated by both the switch and the controller are asynchronous. In this way, neither end must wait for the requested information to return. It is also worth mentioning that Openflow 1.4 was used. (i) **OFPFeaturesRequest** is triggered when a switch joins the controller domain. Its purpose is to collect basic information, such as the switch ID and the amount of existing tables. (ii) **EventOFPSwitchFeatures** is the response event generated by the switch upon receiving the event *OFPFeaturesRequest*. (iii) **EventDP** is sent from the switch to the controller containing information about its local ports (indexes and physical addresses). (iv) **EventLinkAdd** is generated when a new link is added to a switch. This link is necessarily the connection between two switches, since the connection between stations and switches is handled by another event. It is through this event that the link aggregation groups are created. (v) **EventLinkDelete** is the opposite of *EventLinkAdd* and occurs when a link between switches is removed. It is used to remove links from aggregations. (vi) **EventOFPPacketIn**: When the switch does not have enough information in its internal tables to perform the packet switching, it sends this event with the packet information to the controller. It is through this event that link aggregation policies are configured. (vii) **EventOFPErrorMsg** is generated by the controller when there is an incorrect parameter in the

messages exchanged between it and the switch (e.g., creation of a flow was sent to the switch without the destination port information). Used to debug the system.

B. Identification of aggregated links

When the SDN controller receives a notification of the inclusion of a switch in its domain, it checks the links of that switch with its neighboring switches. If more than one link is identified between the same pair, the controller assumes that there is a link aggregation and creates a data structure to store that information. Such structure is created using a dictionary, where the index represents the IDs of the switches and the values represent the interfaces of each switch that are part of the aggregation. With this mechanism, there is no need for manual configuration by a network administrator.

C. Loop prevention mechanism

In network environments, when a message generates a response, that generates a new message, a cascading negative effect is created. Whenever a switch has no information from which to send a particular packet, it sends the packet to all its local interfaces (with the exception of the source interface). If there is more than one interface connected between two switches, at a certain point the packet will be sent back to the switch. Because there is no time-to-live in layer 2 of the network protocol, multiple copies of this packet will be sent and re-sent between the switches, eventually causing a memory overflow and CPU exhaustion of the devices and completely paralyzing the computer network. Known as broadcast storm, this serious problem is prevented using a simple idea: even with all broadcast and multicast packets being forwarded to all ports on the local switch, the switch that receives these packets on ports in an aggregation only processes those arriving in the first port of the group aggregation. This means that if a particular switch has ports 1, 2, and 3 in an aggregation, all broadcast and multicast packets received by ports 2 and 3 are ignored.

D. Fail-safe mode

If a given aggregation interface becomes unavailable for any reason (disconnected cable or disabled interface, for instance), our algorithm have the ability to recognize this failure and to adapt the aggregation group by removing the faulty interface from it. The opposite situation also occurs: if the same interface becomes active again, or if a new interface is added between a pair of switches, the aggregation group is updated with the new interface.

E. LEDE

The LEDE project[1] (Linux Embedded Development Environment) is a Linux operating system based on OpenWrt[2]. Both are used to create custom firmwares from various wireless routers manufacturers. Through these firmwares, in addition to greater flexibility in the equipment configuration, a large ammount of new packages can be installed. It is possible to install Open vSwitch on small home routers.

[1]https://lede-project.org/

[2]https://openwrt.org/

F. Link aggregation algorithms

In SDN networks, we have the freedom to implement our own link aggregation algorithm. During this work, we identified and performed the aggregation using three distinct techniques that will be described next. It is noteworthy that the Mininet emulation platform does not support the configuration of the LACP protocol between two virtual switches. For this reason, we have chosen to implement an algorithm similar to LACP, called Hash Table, to overcome this limitation.

1) Hash Table: The stations connected on the switches start transmitting data. When the switch receives the first flow packet, it re-transmits it to the controller, which identifies the available fields (source and destination MAC address, source and destination IP address) and calculates a hash, that determines which interface will be used to transmit data from that flow. The controller inserts the switching rule into the flow table, which sends the packets of that flow through the interface computed by the controller. Algorithm 1 represents the implementation of this procedure.

Algorithm 1 Hash Policy Algorithm

```
1: function CALCULATE HASH(packet)
2:     value = hash(source mac, destination mac, source ip, destination ip)
3:     return value
4: end function
```

2) Traffic Analysis: The switch, upon receiving the first flow packet, retransmits it to the controller, which identifies the bandwidth utilization of each aggregated interface to make the decision of which interface will be used to transmit the data for that particular flow. The priority is to choose interfaces with low bandwidth usage, allowing a better distribution of traffic between all interfaces. The controller inserts the switching rule into the flow table. Then, the switch forwards the packets from that flow to the interface chosen by the controller. Algorithm 2 represents the implementation of this procedure.

Algorithm 2 Traffic Analysis Policy Algorithm

```
1: function INTERFACE WITH LOWER TRAFFIC(switch)
2:     Read traffic utilization from all interfaces
3:     Identify the interface with lower traffic utilization
4:     return interface
5: end function
```

3) Virtual Round-Robin: The difference between this technique and the previous two ones relies in the amount of rules created. Instead of creating rules for only one flow for each transmission from a source to a destination, the controller creates rules for all sources and all targets using all aggregated interfaces. Yet, each rule is created with different priorities, making the switch choose only one interface to transmit the packets.

However, from time to time, the controller changes the priorities of all rules, forcing the switch to constantly change the interface it will use to transmit the data. During our tests, we have determined the update to happen every 0.2 seconds. This value was determined after performing several tests to identify which would lead to the best Justice Index, which is an indicator used to determine whether users or applications are receiving a fair share of the network bandwidth. In this way, all the interfaces of the aggregation end up being used evenly. Algorithm 3 represents the implementation of this procedure.

Algorithm 3 Virtual Round-Robin Policy Algorithm

```
1: while True do
2:     next_index ←(index+1)%Number_of_rules
3:     Rule[next_index].priority ← 65535;        ▷ Increase priority of next rule
4:     Rule[index].priority ←1;                  ▷ Decrease priority of current rule
5:     index ← next_index;                       ▷ Update index
6:     Sleep τ ms
7: end while
```

IV. Experimental Evaluation and Results

The purpose of this research includes both an experimental and a practical approach. To achieve the first objective, a virtual environment with Mininet, Ryu Framework e OpenFlow 1.3 was created.

The virtual experiment was divided in two separate flows. In the first one, we used the *iperf* tool to generate TCP and UDP traffic among 16 stations in the topology. The *iperf* does not consider the throughput overhead of protocol headers, such as Ethernet, IP, UDP, or TCP. Therefore, in the presented results, the theoretical maximum available bandwidth was not reached. The second set of tests was performed using the *tcpreplay* tool using two realistic traffic files: one captured during a file transfer and another captured during a regular web browsing activity.

A. Experimental Topology

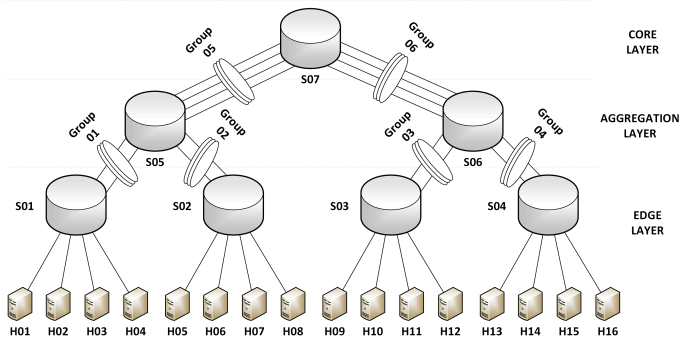


Fig. 1. Lab topology with link aggregation

In order to evaluate and validate our research, a topology inspired in the classic Fat-Tree topology [18] was created. With a total of seven switches and sixteen hosts, the topology consists of three levels: core, edge and aggregation layers. Between each layer, two or more links were connected among the switches to provide better throughput for the data transmission, whereas they were all configured with a speed of 100 Mbps. Figure 1 shows this topology.

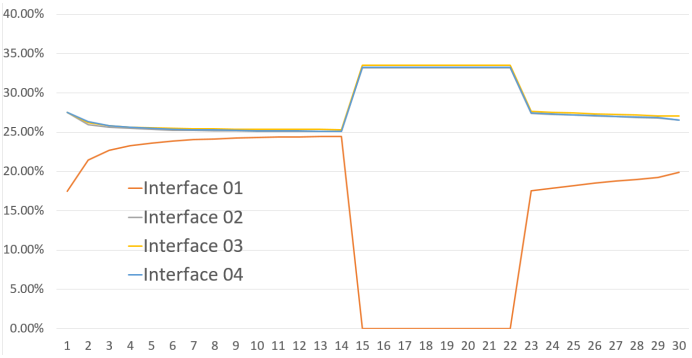


Fig. 2. Interfaces usage during the test

Our virtual environment was built using a virtual machine in Google's cloud platform. Since only a basic configuration was needed, a VM with two Intel 2.30 GHz CPUs, 8 GB of RAM and 10 Gb of disk space was created.

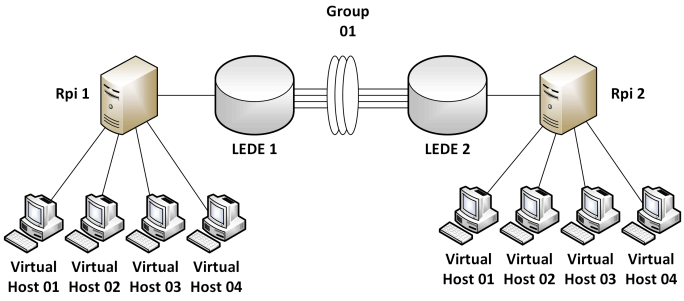


Fig. 3. Real-life topology with link aggregation

To the extend of stressing the virtual topology with the maximum data transmission, the *iperf* tool was chosen to create flows from the first eight hosts to the last eight ones. Both TCP and UDP transmissions were made, and all three algorithms were tested. In this scenario, it is worth to mention that the maximum throughput would be 400 Mbps.

1) Realistic Traffic: On account of *iperf* not being able to simulate realistic traffic scenarios, we chose to use another tool to achieve this goal. The *tcpreplay* tool is able to rebuild and retransmit a given *pcap* file (extension generally created while sniffing a network) from one source to one destination. With this ability, we decided to realize a set of tests simulating an Internet browsing, generating flows from the first eight hosts to the last eight ones. Only the left interfaces of switch 07 were observed during this test, since this is the core switch of the topology and all traffic runs through it. Figure IV-B1 contains the CDF for this test. In this case, the maximum throughput would also be 400 Mbps.

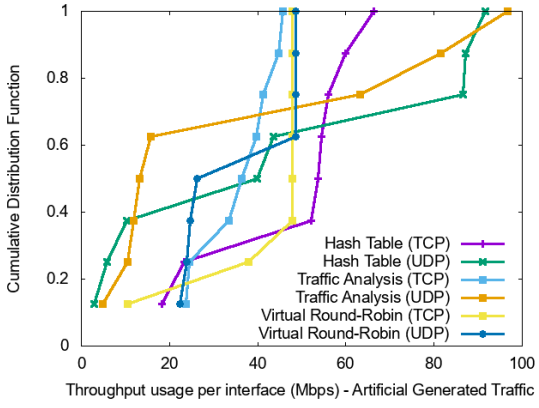
B. Real-life Environment

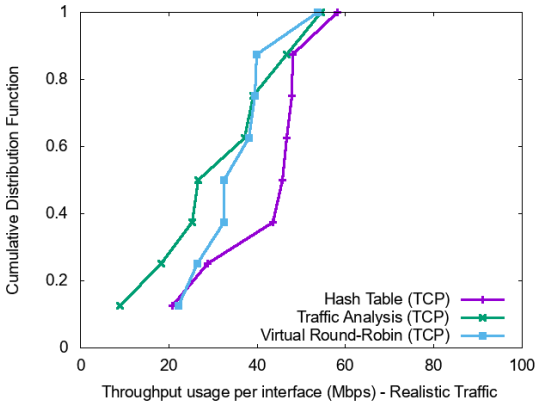
As a proof of concept to demonstrate the real-life application of our research, we were able to execute our algorithms in real equipments. To accomplish this milestone, we setup the following real lab: (i) two low-cost wireless routers with a custom firmware were purchased to create our lab. The hardware chosen was a TP-Link WR841N wireless router, with only 4 Mb of ROM, 32 Mb of RAM and 5 Fast Ethernet ports (one for WAN connectivity and four for LAN connectivity). Due to its memory limitations, a custom firmware was built using the LEDE Project. Only two of these routers were used, connecting all four LAN interfaces between them, creating one single aggregation group. (ii) a Rasperry Pi was used to connect to the switch's WAN port. Raspbian was installed, as well as the latest version of the Ryu controller. (iii) due to the lack of more physical network ports, virtual interfaces were configured in each Raspberry Pi to simulate real hosts connected to both switches. In total, four virtual interfaces were created in each side. Figure 3 depicts the real life topology.

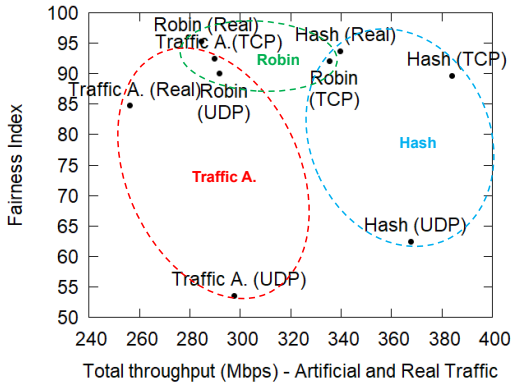
1) Realistic Traffic: Identical to the virtual topology tests, the realistic traffic test performed at the real-life lab uses the *tcpreplay*, as well as the same *pcap* files. Our test consisted in generating flows from the first four hosts to the last four ones, simulating an Internet browsing one more time.

To verify the platform behavior during an interface shutdown, the following test was performed: (i) Using the *iperf* tool, data flows were created between the first four and the last four stations. (ii) Observing the input interfaces load connecting switch 07 to the switch 5, after 15 seconds of transmission, the interface 01 of the aggregation was shutdown. (iii) After 6 seconds, the same interface was reactivated and the test was terminated after 30 seconds of transmission.

Figure 2 displays the distributed traffic load between the four aggregation interfaces. It is notable that while the interface was shutdown, the load was equally distributed among the other aggregated interfaces. When this interface became available again, the traffic was evenly distributed among all aggregated interfaces.





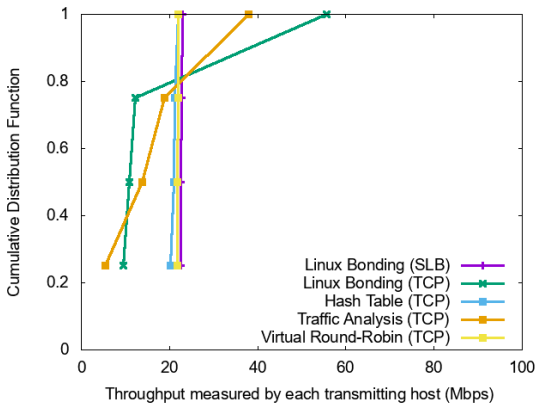


C. Discussion

However, the Traffic Analysis algorithm performed poorly on almost all tests. This means that the implemented code needs to be improved or the policy used is not good to be used in a link aggregation environment. The way the aggregation was chosen at the time of the creation of the flow also needs to be evaluated. Probably, the initial packets are sent to the controller without a considerably increase in the traffic of the links, making the same interface (or the same set of interfaces within the aggregation) to be selected at the beginning of the flows. In any case, further analysis needs to be performed using packet capture, thus optimizations in the algorithm code need to be made.

It is definitely noticeable the high accuracy that our approach had in the real-life lab, against Linux's native Ethernet Bonding. Both our Hash Table and Virtual Round-Robin algorithms were able to divide traffic equally among all interfaces in the local aggregation.

To elucidate the results obtained, we created fours graphs: (i) Figures IV-B1 and IV-B1 represent the cumulative distribution function (CDF) for the virtual environment. In both of them, the area between 40 Mbps and 60 Mbps represents the best balanced distribution. (ii) Figure IV-B1 depicts the scatter plot of fairness per throughput for all policy algorithms in the same environment. (iii) Figure IV-B1 represents the CDF for the real-life environment. In this one, the area between 20 Mbps and 30 Mbps represents the best balanced



distribution.

Despite the difference of balanced traffic in the virtual environment, it can be noticed that our dynamic approach had a great performance compared to native and well-known aggregation protocols, such as Linux Ethernet Bonding.

V. Conclusion and Future Work

The results obtained in this work show that the proposed new approach for link aggregation in SDN networks presents a better way to deal with the classical link aggregation administration. The dynamism, complexity and scalability of current network environments justify the need to find a new solution to the old bottleneck problem, an that is the main contribution presented here.

The decision to modify the packet path from time to time was presented as the best way of balancing the traffic between all the interfaces of an aggregation, considering all the policies discussed. This premise made the distribution fairer and the results presented in this work approximate the results obtained using a solution widely adopted in real environments. In addition, the presented solution was able to use 84% of the available bandwidth during the TCP traffic and 91,4% during the UDP traffic. The fairness index was also very satisfactory, 94% in TCP traffic and 98% in UDP traffic.

As future work, the integration of a system with a traffic prediction tool to enable real-time traffic engineering is envisaged. Also, the complete implementation of the dynamic and self-adaptive solution for intra-flow traffic needs to be performed. It is also necessary to study a way to improve the implementation of the Traffic Analysis algorithm, since the way this policy was implemented did not bring satisfactory results for the link aggregation. Additionally, larger tests needs to be performed to analyze the impact of having aggregated interfaces with different speeds. Furthermore, the overhead in the network as the number of flows grows needs to be to investigated when using the Virtual Round-Robin algorithm. Finally, the impact of rewriting flows from time to time needs to be evaluated, since this can be the cause of eventual packet transmission delays.

References

[1] M. J. Ranum, K. Landfield, M. Stolarchuk, M. Sienkiewicz, A. Lambeth, and E. Wall, "Implementing a generalized tool for network monitoring," *Information Security Technical Report*, vol. 3, pp. 53–64, 1998.

[2] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 5, pp. 81–94, 2007.

[3] R. L. Carter and M. E. Crovella, "Measuring bottleneck link speed in packet-switched networks," *Performance evaluation*, vol. 27, pp. 297–318, 1996.

[4] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, Aug. 1995. [Online]. Available: http://dx.doi.org/10.1109/90.413212

[5] M. Seaman, "Link aggregation control protocol," *IEEE http://grouper.ieee. org/groups/802/3/ad/public/mar99/seaman*, vol. 1, p. 0399, 1999.

[6] D. F. Macedo, D. Guedes, L. F. M. Vieira, M. A. M. Vieira, and M. Nogueira, "Programmable networks: From software-defined radio to software-defined networking," *IEEE Communications Surveys and Tutorials*, 2015.

[7] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[8] D. Guedes, L. F. M. Vieira, M. A. M. Vieira, H. Rodrigues, and R. V. Nunes, "Redes Definidas por Software: uma abordagem sistłmica para o desenvolvimento de pesquisas em Redes de Computadores," SBRC 2012, 2012.

[9] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer," in *Hotnets*, 2009.

[10] T. D. Davis, W. Tarreau, C. Gavrilov, C. N. Tindel, J. Girouard, and J. Vosburgh, "Linux ethernet bonding driver howto," *Digital Equipment Corporation. Tech. rep., Technical Report DEC-TR-301*, 2011. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/bonding.txt

[11] M. Bredel, Z. Bozakov, A. Barczyk, and H. Newman, "Flow-based load balancing in multipathed layer-2 networks using openflow and multipath-tcp," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 213–214. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620770

[12] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: topology, routing, and packaging of efficient large-scale networks," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Nov 2009, pp. 1–11.

[13] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855730

[14] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Dctcp: Efficient packet transport for the commoditized data center," *Digital Equipment Corporation. Tech. rep., Technical Report DEC-TR-301*, 01 2010.

[15] A. Al-Shabibi, M. D. Leenheer, M. Gerola, A. Koshibe, G. M. Parulkar, E. Salvadori, and B. Snow, "Openvirtex: make your virtual sdns programmable." in *HotSDN*, A. Akella and A. G. Greenberg, Eds. ACM, 2014, pp. 25–30. [Online]. Available: http://dblp.uni-trier.de/db/conf/sigcomm/hotsdn2014.html#Al-ShabibiLGKPS14

[16] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 266–277. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018467

[17] R. D. Vencioneck, G. Vassoler, M. Martinello, M. R. N. Ribeiro, and C. Marcondes, "Flexforward: Enabling an sdn manageable forwarding engine in open vswitch," *10th International Conference on Network and Service Management (CNSM) and Workshop*, pp. 296–299, 2014.

[18] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, vol. C-34, no. 10, pp. 892–901, Oct 1985.