ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 6, November 2015

An Approach for Modeling of Traffic Engineering for Software Defined Network Data Paths

Lucio Agostinho Rocha

***Abstract*— SDN is a paradigm that proposes the separation of data forwarding plane from the data control plane. OpenFlow is an standard protocol used in SDN for establishing communication among switches and controllers. SDN is one of the most promising approaches to the deployment of future network infrastructures. The most of the Internet service providers have to deal with a number of configurations to a crescent amount of network devices. We propose evaluate the feasibility of deployment of data paths supported by computational modeling in emulated scenarios using our open source emulator known as MILP Flow. Our aim is argue about the many issues in the deployment of data paths that are to be set before data flows are sent through the network. As a consequence, MILP Flow contributes as an case study to reduce the overhead to discover network routes in large network environments.**

***Index Terms*— SDN, Open Flow, Network Management.**

I. INTRODUCTION

In conventional data center networks the performance is often determined by the efficiency of the data transportation inside the data center [15–16]. In this sense, the utilization of SDN networks brings new alternatives to deal with routing inside of data centers. SDN is a network architecture that proposes the separation of data forwarding plane from the data control plane. In particular, the network devices are seen as simple forwarding elements and the common management of the data control plane is implemented in a separated SDN controller out of these network devices. SDN architectures commonly use the Open Flow protocol [17–18] to set rules in the data plane. Computational modeling for Software Defined Networks (SDN) is a recent and challenging research area. For the best of our knowledge are still few works in the literature that integrates computational modeling with the deployment of data paths for SDN networks. We aim that this integration is important to evaluate the traffic demand before prototyping. This is relevant in many scenarios where experimentation in real SDN networks is difficult due to the complexity of management, time for deployment and capital expenditure to evaluate data traffic.

The most of SDN controllers not impose restrictions about how many flows will be forwarded by its network devices. Despite the fact that a solution “de facto” is still not confirmed in the literature, many SDN controllers are been developed in different high-level languages. For high amount of data traffic is necessary an accomplish between high level solutions for management, extension, and development of SDN applications, and low level solutions to forward big amount of data traffic in a reasonable way that be competitive in terms of quality of services[2], such as throughput, drop of packets, delay, jitter and other parameters commonly accepted in conventional networks. SDN solutions are still very restricted in terms of API standardization, common libraries, and even the way of establish data paths in non-trivial topologies. Also, debugging and troubleshooting are employed of different ways by each vendor. Even with these aspects is still possible argue about methodologies to support large controlled network by OpenFlow enabled devices.

Computational modeling [3] is crucial for network management [1] [4–8]. We follow this assumption in the development MILPFlow[14] (Mixed Integer Linear Programming with OpenFlow): a toolset for computational modeling integrated at deployment of SDN networks using the OpenFlow protocol. Our approach is use MILP modeling jointly with the deployment of data paths in the SDN network. We validate our approach using the conventional Mininet emulator. MILPFlow is used to generate MILP models about the data center resources, map the solution in OpenFlow rules and deploy these rules in data paths of SDN networks. This whole mapping is important to simplify the management of OpenFlow rules by data center administrators for large SDN topologies. The remains of this paper is structured as follows. Section II is about the main related works considered in our research; Section III presents the requirements for integrating computational modeling with the management of

OpenFlow rules. Section IV is about the MILPFlow software implementation. Section V presents evaluations with different protocols to forward data traffic; Fynally, we done final considerations in Section VI.

II. RELATED RESEARCH

In this section we detailed the related research about management of data paths in SDN networks. RouteFlow [19] is an architecture that follows the SDN paradigm and provides a way to integrate many routing control platforms. The physical topology is mirrored through Virtual Switches (VS) that are distributed in Virtual Machines (VM) controlled by the platform. Inside each VM is running an own routing control platform, such as Quagga, and each VS (lower level) is able to reach each other because these VMs (high level) are also connected to each other. Sharma et al. [20] proposes automatic configuration of this environment in reduced time using a GUI (Graphical User Interface), but this attempt is still done with a few switches (a pan European topology of 28 switches). The authors use different controllers to acquire topology data and to share the automatic load configuration of its environment. A proof-of-concept of Networking Function Virtualization (NFV) to perform routing is studied by Batalle et al. [21].

The authors follow the concepts of NFV to guide decisions out-of the-box of network devices. They conclude that it is possible to move the functions of routing along the network duplicate it and keep dynamic routing tables between the different network elements. One of the benefits is the centralized management of the routing with reducing of the effort to configure its devices. EstiNet [22] is an OpenFlow network simulator and emulator. It combines the advantages of both these approaches to run tests with very large number of OpenFlow switches and hosts. It uses a real OpenFlow controller, network applications and the real TCP/IP protocol stack in the Linux kernel. The authors say that EstiNet is more scalable than Mininet, and its performance with an OpenFlow controller are also more accurate than Mininet. Nevertheless, this software tool is not freely available as an open source project. The most popular network simulator in the literature is the ns–3 [23]. This network simulator use its own OpenFlow module written in C++. To run simulations with this module, it is necessary to compile it and link it jointly with the network simulation engine. Currently, ns–3 no offer a spanning tree protocol or interaction with real OpenFlow controllers, and its OpenFlow support it is in early stage of development.

Ripcord [24] is a library that provides a framework to simulate data centers topologies with OpenFlow controllers in Mininet. The purpose of this library is evaluating in software multiple routing applications with different schemes of commonly used data center topologies, such as Fat-tree, VL2 and Portland. However, it is a hard task to analyze the data paths obtained by this library. Also, we observe that Mininet runs on a single host to emulate whole its links, networks devices and hosts. These instantiated resources share the computational resources of the physical host that runs Mininet, what difficulties the evaluation of large scale experiments. As an alternative, the toolkit DOT [25] distribute the emulated SDN network using multiple physical machines to provide resources for its network components. The toolkit uses an embedded algorithm that minimizes the physical bandwidth load and the number of physical machines need to support the specified network topology. Also, is possible to perform the emulation using DOT inside VMs with Qemu emulator. Another alternative is use the toolkit Maxinet [26] that span the emulation of larger topologies across physical machines. All switches are controlled by one central OpenFlow controller that regulates in a transparent way the routing between its worker nodes that are the physical machines that instantiate the Mininet switches. However, it is not a trivial task to map where these switches are deployed, what difficulties the direct deployment of OpenFlow rules. We note that SDN/OpenFlow experiments with computational modeling are not present in any of the proposals presented above. Our work is innovative in the sense that we aim at contributing to the state of the art in affordable yet rich SDN experimentation using computational modeling jointly with the deployment of rules in the network.

III. METHODOLOGY

In this section, we first formulate the key requirements to enable routing in SDN networks and then describe our technical approach. Although this list is not exhaustive, it is necessary to the most of the SDN scenarios that use the OpenFlow protocol for routing.

A. Requirements

We argue in **R1** the requirement to define a modular component with a set of routing functions. Although it is difficult to provide modular components interchangeable among different controllers, an alternative is to insert these functions in a module of the OpenFlow controller. We illustrate these scenarios in Fig. 1: a) The internal module for routing analyzes packets and send routing instructions to the SDN network. This requirement defines a protocol to allow the exchange of routing functions with the controller; b) The second alternative is to utilize an external module with the indirect deployment of rules. The routing is indirect because it still depends on the controller to send rules to the switches. This approach must define a secure protocol between the external module, deployed on a different machine, and the controller. c) The third alternative is to use an external module with the direct deployment of rules without the presence of an OpenFlow controller. In this case, it is possible to send forwarding rules directly via dpctl commands to the flow tables of the switches; **R2** is the requirement to establishment of data paths should be done in not more than a few seconds. This implies that the configuration of routing rules should be near at real time; **R3** is the requirement about that routing should be easy to configure in a few steps to allow automated tasks during critical periods such as temporary peak of load in links, fault in devices, addition/remove of devices, and others; **R4** is the requirement about routing feasibility before setting the OpenFlow rules in the network devices. This requirement is necessary to avoid including a lot of unused rules in switch tables; **R5** is the requirement to establishment of new data paths should consider the previous set of routes, and the current link loads. This is necessary to avoid routes over congestionated links. These requirements were considered as a guide to define our methodology. Other issue is about the method to deploy the routing rules in the network. Most common alternatives are the reactive and the proactive methods that are explained as follow.

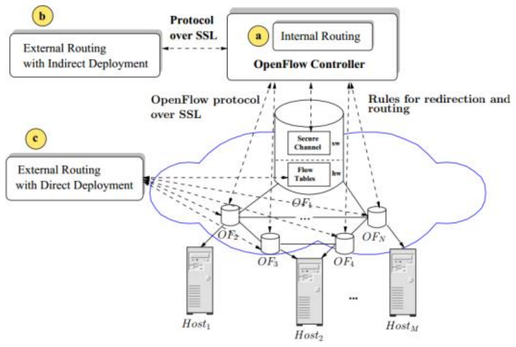
b
Protocol over SSL
a
Internal Routing
External Routing with Indirect Deployment
OpenFlow Controller
OpenFlow protocol over SSL
Rules for redirection and routing
c
Secure Channel
sw
External Routing with Direct Deployment
Flow Tables
hw
OF_1
OF_2
OF_N
OF_3
OF_4
$Host_1$
$Host_2$
$Host_M$

Fig.1 – Approaches for Routing in SDN networks.

B. Reactive vs. Proactive routing in SDN

When a packet is received, the OpenFlow switch tries to match the header of this packet in its flow table. If no information about this packet is found, the switch sends the headers to its OpenFlow controller. This will cause a packet in event in the controller. Then, the controller will take some actions considering the headers of the L2, L3, and L4 so that this packet and further ones belonging to the same flow are sent throughout the network. When a route is obtained by the controller, flow mod messages are sent back to the switches to set their flow tables. We illustrate this process in Fig. 2. The establishment of these data paths is done with proactive or reactive methods.

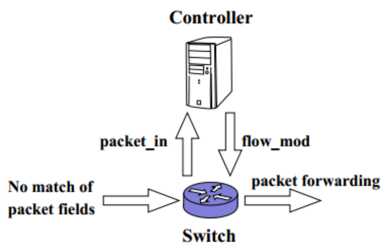

Fig. 2 – OpenFlow Packet Forwarding

Proactive routing methods in SDN networks assure that paths are set before packets are sent in the network. The advantage is that they do not generate additional network overhead and do not affect the network performance to discover its end hosts. When packets get in the network, the forwarding rules are already there and no packets are sent to the controller. However, proactive methods depend on previous mapping of the whole interconnections in the network topology, which is not always feasible for all networks. Reactive routing methods occur when new a packet reaches a switch without a matching in the flow entry. Then, such packet must go to the controller which in

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 6, November 2015

turn uses control messages to automatically discover its end-hosts and obtain the active links before the data traffic be forwarded in the network. The advantage is that there is no need of previous knowledge of the whole network topology. However, for large networks, the amount of control messages will potentially influence the network performance since all the first packets of every new flow will go to the controller before traversing the network.

Both proactive and reactive methods are useful to set the paths in SDN networks. Proactive methods require synchronization between the controller that deploy paths. Reactive methods introduce additional overhead in large networks and influence the network performance. Reactive flow setups add latency to the first packet of a flow, but the subsequent flow forwarding state is cached on the OpenFlow switch so that this process is not done for other packets of the same flow.

C. MILPFlow Methodology

Firstly, we are conducting our research to increase the scalability. We are currently improving our MILP model to contemplate MILP and heuristics to solve data paths in larger topologies. Currently we focus on the evaluation of the aggregated traffic between ToRs, not between virtual machines (VMs). Our MILP models not contemplate the modeling of packet losses. Indeed, our modeling is about the establishment of concurrent data paths without losses. So, the evaluation section shows that our MILP model is feasible to forward many concurrent flows without losses. Fig. 3 shows that our methodology is not a circle, but the correct interpretation is that the whole figure is the methodology.

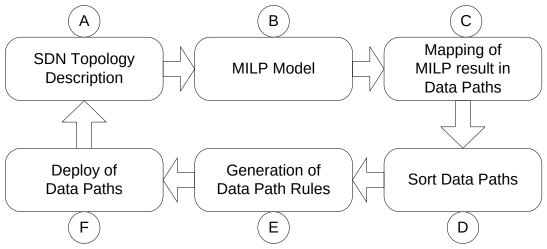

Fig. 3 – MILPFlow Methodology

Fig. 3 illustrates the MILPFlow methodology. Our intent is to keep the main functions of the OpenFlow controller to deploy data paths for large SDN topologies, avoiding the increasing of control messages in the network. The proposed method is similar at Multi Commodity Flow (MCF) [9–13] problems. Moreover, our modeling at this time is not based on heuristics, but in mathematical feasibility of MILP problems. We emphasize the explanation about the methodology to join MILP and SDN modeling, more than specific mathematical details. The focus of this current work is on showing a management tool which works together with modeling and deployment.

A. SDN Topology description

Describes the network characteristics of the SDN environment in terms of network topology, traffic matrices, racks of servers, number of servers, number of switches, network links, bandwidth, delay, and other customized information. In particular, we describe our traffic matrix as the relationship of the whole data traffic that is forwarded from each source Top of Rack (ToR) until reach its destination ToR. The values of these matrices are well known inside the datacenter or can be given from statistics and/or historical data about the load of communication.

B. MILP model

It is used to map the SDN topology description in a MILP problem. We describe the details of this formalization in [9] where we define a linear model for VM placement. However, in this paper, we extend this model to establish data paths to SDN topologies. In our model, these elements are ToRs, switches, network communication links, and traffic matrices. The processing of MILP model is done with MILP solvers, such as CPLEX or Lingo solvers. The result obtained from solving this MILP model is a set of routes to forward data among ToRs, without over-utilization of the network links.

The internal network is modeled as a directed graph where the vertices are network nodes (core switches, access witches and ToRs) and edges are communication links connecting the nodes. We consider the following types of nodes:

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 6, November 2015

1) Source nodes: ToRs generating network traffic;
2) Sink nodes: ToRs consuming network traffic;
3) Topology nodes: topology nodes responsible for routing network traffic.

Nodes are modeled as a set of incoming and outgoing network interfaces. A link $L_{p,q}$ connects the network nodes p and q and defines pre-configured paths for transporting network flows. In the MPLS (Multiprotocol Label Switching) technology [27], such paths are potential constraint-based routes, or LSPs (Label Switching Paths). Links have a maximum capacity (bandwidth) represented as $L_{p,q}^{bw}$. Each link $L_{p,q}$ carries a flow $F_{p,q}$. Let us define:

- F_p^{source}: flow generated by the source node p
- $F_q^{\sin k}$: flow consumed by the sink node q.

We use the description of the aggregated of traffic among ToRs. Our intent is to deal with the flows that potentially will impact on the network performance. Flow conservation assures that the sum of flows generated by all the sources considering uplinks and downlinks is equal to the sum of flows consumed by all the sinks. This same description is valid for all the switches in the network:

$$\sum_P F_p^{source} - \sum_q F_q^{\sin k} = 0, \forall p, q \qquad (1)$$

Also, flows cannot exceed the capacity of the links transporting them:

$$F_{p,q} \le L_{p,q}^{bw}, \forall p, q \qquad (2)$$

Once the restrictions have been established, we can set the objective function that minimizes the total cost C_j to forward traffic among the ToRs, according to the pre-defined traffic matrix among these ToRs. With the latter restrictions, the problem is stated as follows:

$$\min \sum C_j + \sum traffic(ToR_p, ToR_q), \forall p, q \qquad (3)$$

Equation (3) is the objective function of our modeling. Indeed, our approach reduces the computational complexity and keep it lesser than O(n^2). This is a minimization problem because we want to acquire the best routing according to the current network load without losses in links. The solution of this problem provides the near optimal paths required to forward the whole traffic specified in the SDN topology description without violating the capacities of the network links. The remaining servers and network elements that are not used to forward traffic are suitable to be put in low power mode or even be turned off. Using a MILP model, we satisfy R3 because many configurations are feasible to be evaluated by changing inputs of a same representative model. R4 is satisfied because we evaluate the feasibility of the inputs before the deployment of switches in SDN network. R5 is satisfied because our model allows to obtains effective routing of many concurrent flows at the same time, and set specific flow mod rules for each one.

C. Mapping of MILP Result in Data Paths

The result obtained from solving this MILP model is a set of routes to forward data among ToRs, without over-utilization of the network links. We show an example in Fig. 4 and its correspondence mapping of MILPFlow results in Table I. The data structure *F1* keeps the whole information about the links with flows of *F1*. So, *F1[1][2] = 1000* indicates that 1000 units of the flow *F1* goes through the link between the nodes H_1 and s_2. Our goal is simplify the mapping of MILP results on data paths.

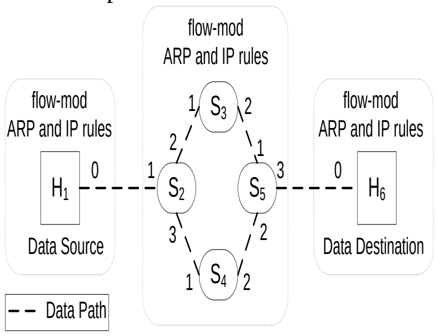

Fig. 4 – Mapping of Data Paths with MILPFlow.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 6, November 2015

TABLE I. MAPPING OF PORTS WITH MILPFLOW

MILPFlow results	Mininet Topology	MILPFlow Ports
F1[1][2]=1000	(h1,s2)	(h1,0,s2,1)
F1[2][3]=200	(s2,s3)	(s2,2,s3,1)
F1[2][4]=800	(s2,s4)	(s2,3,s4,1)
F1[3][5]=200	(s3,s5)	(s3,2,s5,1)
F1[4][5]=800	(s4,s5)	(s4,3,s5,2)
F1[5][6]=1000	(h6,s5)	(h6,0,s5,3)

D. Sort of Data Paths

After executing the earlier step, an extra effort is necessary to compose the data paths. As shows the Table I, the flow F1 occurs on many links. However, it is necessary to compose the path in the properly sequence, i.e., the sequence of nodes to forward data from source ToR to reaches its destination ToR. For this we employ an algorithm similar at the depth-first search algorithm in to explore as far as possible each branch in the path of *F1*. We illustrate an example in Fig. 4. In this example, the sequence *F1[2][3]* and *F1[2][4]* are the links between the node s_2 to reach the nodes s_3 and s_4. So, we have a split condition in this path. Similarly, the sequence *F1[3][5]* and *F1[4][5]* are a join condition near at the destination.

E. Generation of Data Path Rules

We describe this step with an example of a flow generated from host H_1 to host H_6, as illustrated in Fig. 4. Each switch s_i communicates with each other via communication ports. We combine MILP results with Mininet to create a mapping of MILPFlow ports, as shown in Table I. The first column in Table I shows the route created by MILPFlow to go from host H_1 to host H_6. The second column shows the hosts and switches in the Mininet Topology and finally in the last column, we see each host-port-switchPort connection. Emulators as Mininet read topology files to create its virtual connections with Open vSwitch (OVS) kernel switches. Mapping the input and output ports of each link that connects switches is necessary to establish connectivity hop-by-hop, and this task is generally performed by OpenFlow controller. However, we acquire this information directly from Mininet (parsed from *net* Mininet command). As a consequence, it is not necessary to send control messages to discover the connection between the ports of these switches. However, it is necessary to define ARP and IP flow mod rules for the switches of each data path.

When splits of traffic occur we employ the group tables of the OpenFlow 1.1.0 specification. This feature allows create the split of flows in the network as well as their subsequent joins. For each group table, the value of the variable weight indicates the amount of traffic that should be forwarded through its sub-paths, as occurs in the switches s_2 and s_5 in Fig. 4. We set the normalized weight value of the MILP result. As an example, we map the resultant flow of 200 units in the link between the switches s_2 and s_3: *F 1[2][3] = 200*, and between the switches s_2 and s_4: *F1[2][4] = 800*. Then, we set *weight1=2*, *weight2=8*, respectively.

F. Deployment of Data Path Rules

MILPFlow generates the set of flow mod and group mod rules in an executable batch file. The network administrator uses these rules to deploy its data paths. Two output formats are generated: *dpctl* commands and HTTP REST commands for using with REST API provided by some controllers, as shown the Fig. 5.

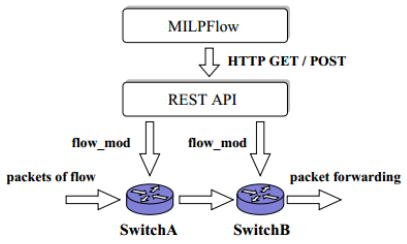

Fig. 5. MILPFlow: Deployment of Data Paths.

IV. IMPLEMENTATION OF MILPFLOW COMPONENTS

MILPFlow is a project written in the Java programming language. Fig. 6 shows its components that are implemented as classes. Our intent is to provide an open source toolkit that is easily extensible. These components are described in UML (Unified Modeling Language) as follows:

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 6, November 2015

• MILPFlow Engine is the main component that controls the interactions with all other components;
• GraphViz Engine is the component that creates the graphic files to visualize the generated data paths;
• Topology Descriptor is the component that reads the variables of the simulation, such as topology nodes, links, bandwidth, number of servers, number of switches and traffic matrix;
• GeneratorMILP is the component that produces the MILP model to be solved with a MILP solver;
• MILPSolver is the component that invokes the MILP solver;
• MininetTopology is the component that performs the mapping between the MILP results and Mininet network;
• OpenFlowRules is the component that produces the executable batch file with dpctl commands and/or HTTP REST commands from the MILP results.

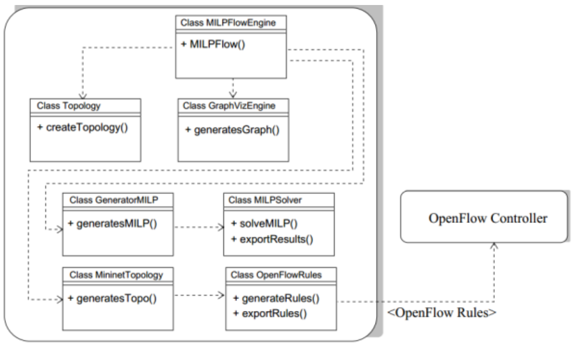

Fig. 6 – MILPFlow Components

The implementation offers an approach to gain input parameters for fine-grained Traffic Engineering (TE). From a specified topology description, the user defines the estimates about the maximum load between its hosts, and MILPFlow generates its correspondent data paths. It is interesting to note that all data path rules are set at the same time, i.e., exist a strict relationship between these rules to avoid conflicts when all of them were deployed in the SDN network. So, MILPFlow implementation pre-calculates paths for all concurrent flows. The software is independent of OpenFlow controller when generates *dpctl* commands, although also generates HTTP REST requests specific to REST APIs.

V. EXPERIMENTAL EVALUATION

The purpose of the evaluation is to show the feasibility of the tool in terms of mathematical modeling and the deployment of flows in a real network. In order to show that MILPFlow keeps the same features as other well known protocols, we compare it with the Spanning Tree Protocol (STP) implementation of the Ryu OpenFlow Controller. Our intention is evaluate the throughput of the MILPFlow data paths. However, this evaluation has not intention of comparing MILPFlow and STP in terms of offering the best throughput for the flows. We perform the comparison with MILPFlow and Ryu STP protocol. Also, we use Iperf with all the measurements between the hosts at the same time. This is shown in the topology (logical links). Also, the flow are submitted at the same time, and the correct interpretation of the results is not possible without considering the expected concurrency of these flows. We are currently considering improving our work with the SDN modeling for larger topologies in multi-domains. The purpose of the RTSP is to show a real application that runs with our methodology, and we observe results closer at well-known protocols such as STP.

We evaluate a set of scenarios on a Dell PowerEdge R420 with Intel Xeon Quad Core, 2.4GHz, and 48GB of RAM. Our experiments run inside VMs of Virtual Box with XUbuntu 12.04.2 LTS with 1 Gbps NICs, and Open vSwitch compatibleOpenFlow switches. We use Mininet and the Fat-tree topology of Fig. 3. Our experiments were done with Iperf software, and streaming of video using the VLC application with Real-time Streaming Protocol (RTSP). We choose the Ryu controller since it provides a REST API and offers high performance to establish data paths for non-trivial network topologies with loops in its structure (e.g., fat-tree, BCube, VL2 topologies). In order to show that MILPFlow keeps the same features as other well known protocols, we compare it with the Spanning Tree Protocol (STP) implementation of the Ryu OpenFlow Controller. Our intention is evaluate the throughput of the MILPFlow data paths. However, this evaluation has not intention of comparing MILPFlow and STP in terms of offering the best throughput for the flows.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 6, November 2015

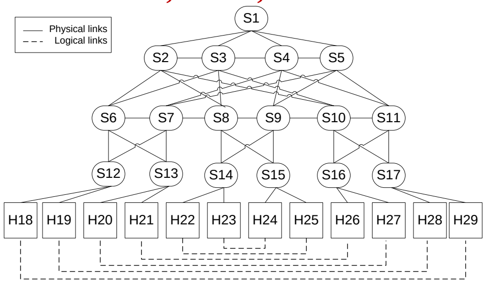

Fig. 7. Fat-tree Topology used in the Experiments.

We use MILPflow to generate the data paths, following the steps of the methodology illustrated in the Fig. 3. The SDN topology description is done for the reference topology of the Fig. 7. This is a fat-tree topology with 29 nodes (17 switches and 12 ToRs) and 36 edges. Our traffic matrix is set to evaluate the throughput between the logical links between the ToRs, and that are showed in dashed lines of the Fig. 7. In our traffic matrix each entry represents the aggregate of the whole flows of traffic that are generated by a source ToR, and that is consumed by its destination ToR. For this experiment we set 6 aggregated flows of 5000 units each to transverse our reference topology. We define the bandwidth of the links in 10000 units to have a value near at 10Mbps of our Mininet bandwidth. The next steps of our methodology are automatically done with MIPLFlow: the generation of the MILP model, the processing of the MILP model, the composing of data paths, and the generation of data paths rules. The output is an executable batch file with the OpenFlow rules to establish the data paths.

The evaluation was done taking into account UDP and TCP traffic. In both experiments we run Iperf measurements 30 times for each pair of ToRs of the logical links. We collect data every 5 seconds for each measurement. For UDP traffic we use Iperf to submit 5Mbps between the ToRs. For TCP traffic we run similarly, but without restriction about the amount of data to be forwarded among the ToRs. The toolkit to perform these measurements is available in our github project.

We compare MILPFlow versus the STP protocol for UDP traffic. We observe that the throughput of MILPFlow is similar to the throughput of the STP proving that the deployment of the MILPFlow routes was well done and is working. The packet losses are below 1% in both scenarios. We observe that the MILPFlow values are near at 5Mbps, what indicates that the results are near at the defined in the SDN topology description. We show a summary of these results in the Table II.

Fig. 9 shows the evaluation done with TCP traffic. The results are similar to the UDP traffic since the throughput is quite the same between MILPFlow and STP and the packet losses are bellow 1%. The exact numbers are shown in Table II. The throughput reached by the STP is a little better, and near at 90Mbps when compared to the throughput reached by MILPFlow that is near at 80Mbps. This is explained since the routes created by STP are always the shortest ones among every pair of servers in the datacenter. This is not true for the routes created by MILPFlow modelling because it accommodates the flows taking into account the capacity of each link and the shortest path is not always obtained for all the flows. We show a summary of these results in the Table II.

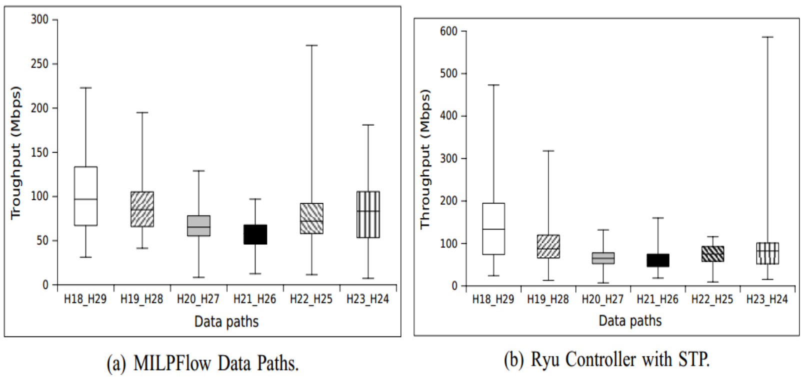

Fig. 9. Evaluation of TCP Traffic

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 6, November 2015

TABLE II. COMPARATIVE EVALUATION WITH IPERF

	UDP		TCP	
	MILPFlow	Ryu STP	MILPFlow	Ryu STP
Average Throughput (Mbps)	4.13	4.25	79.71	89.34
Max. Throughput (Mbps)	4.64	4.69	271	586
Min. Throughput (Mbps)	2.96	3.95	7.34	7.28
Min. Std. Dev.	0.11	0.06	16.34	21.86

We also did an experiment with a real application using RTSP. For this experiment, we choose an H.264 encoded a movie of 10 minutes that streams from the server H18 to the client H29. We perform measurements of throughput with Tcpstat, taking samples at each 5 seconds. The losses of packets bellow 1% indicates that most of the data traffic is forwarded between the hosts. Table III shows that the values of the measurements (M1) with MILPFlow and measurements (M2) with Ryu STP are closer, what indicates a similar behavior in the utilization of both approaches. The standard deviation is high in both approaches. This inaccuracy is mainly introduced by the lack of synchronization between the client and server stream applications. This occurs because we are unable to synchronize the start of these applications, and the traffic measure taken in one given period in server will be taken some seconds ahead in the remote client application. The low percentage of losses gives a good indication of the quality of the data paths established by MILPFlow and the STP of the Ryu controller.

TABLE III. COMPARATIVE EVALUATION WITH RTSP

	MilpFlow	Ryu STP	M1-M2	M1-M2 (%)
Average Throughput (Mbps)	379.36	359.10	20.26	1.06
Max. Throughput (Mbps)	1036.32	971.89	64.44	1.07
Min. Throughput (Mbps)	157.99	144.81	13.18	1.09
Min. Std. Dev.	213.46	200.61	12.85	1.06

VI. CONCLUSION AND FUTURE REMARKS

SDN is an important research area due its independence of proprietary or legacy hardware. Many efforts have been done to increase its scalability to large computational environments, such as data centers. However, many conventional SDN software's are still in early stages of development what difficult the manageability of large amount of routes to forward data traffic. Conventional alternatives that employ dynamic discovery of data paths for traffic engineering generates high overhead in the network due excessive broadcast messages to reach its end points. We use MILPFlow methodology as an alternative to reduce significativelly this amount of data traffic. Our methodology follows a bottom-up approach, i.e., we evaluate and establish data paths in the network before submit any data traffic to discovery routes. We argue that conventional SDN software follow a top-down approach, i.e., they use dynamic data path discovery techniques without description about its network topology. We believe that well-described SDN techniques can potentially reduce the network energy consumption to forward data traffic in large computational environments.

Our future research has the goal of increase the scalability of our approach in a sense of employ techniques of graph theory to reduce the many groups of well-described large topologies. Also, the inherent dynamic of data paths should be considered to establishment of new routes with lesser energy consumption.

REFERENCES

[1] S. Chowdhury, M. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in Network Operations and Management Symposium (NOMS), 2014 IEEE, May 2014, pp. 1–9.

[2] A. Schaeffer-Filho, A. Mauthe, D. Hutchison, P. Smith, Y. Yu, and M. Fry, "Preset: A toolset for the evaluation of network resilience strategies," in Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, May 2013, pp. 202–209.

[3] J. W. Chinnecke, Practical Optimization: A Gentle Introduction, 2012. [Online]. Available: http://www.sce.carleton.ca/faculty;chinneck/po.html

[4] M. Portnoy, Virtualization Essentials. John Wiley & Sons, 2012.

[5] M. Alicherry and T. V. Lakshman, "Network aware resource allocation in distributed clouds," in INFOCOM, 2012 Proceedings IEEE, March 2012, pp. 963–971.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 4, Issue 6, November 2015

[6] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera, “A stable network-aware vm placement for cloud systems,” in Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on, May 2012, pp. 498–506.

[7] G. Wu, M. Tang, Y. Tian, and et al., Energy-Efficient Virtual Machine Placement in Data Centers by Genetic Algorithm. Neural Information Processing - Lecture Notes in Computer Science. Springer, 2012.

[8] X. Meng, V. Pappas, and L. Zhang, “Improving the scalability of data center networks with traffic-aware virtual machine placement,” in INFOCOM, 2010 Proceedings IEEE, March 2010, pp. 1–9.

[9] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, ElasticTree: Saving energy in data center networks, in Proc. 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10), San Jose, USA, Apr. 2830, 2010, pp. 116.

[10] T. Benson, A. Anand, A. Akella, and M. Zhang, MicroTE: Fine grained traffic engineering for data centers, in Proc. 7th International Conference on emerging Networking Experiments and Technologies (CoNEXT 11), Tokyo, Japan, Dec. 69, 2011, pp. 112.

[11] M.Al-Fares, S.Radhakrishnan, B.Raghavan, N.Huang, and A.Vahdat, Hedera: Dynamic flow scheduling for data center networks, in Proc. 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10), San Jose, USA, Apr. 2830, 2010, pp. 115.

[12] Y. Li and D. Pan, OpenFlow based load balancing for Fat-Tree networks with multipath support, in Proc. 12th IEEE International Conference on Communications (ICC 13), Budapest, Hungary, Jun. 913, 2013, pp. 15.

[13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, and J. Zolla, B4: Expe- rience with a globally-deployed software defined wan, in Proc. ACM SIGCOMM 13, Hong Kong, China, Aug. 1216, 2013, pp. 314.

[14] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, Achieving high utilization with software-driven wan, in Proc. ACM SIGCOMM 13, Hong Kong, China, Aug. 1216, 2013, pp. 1526.

[15] MILPFlow. Available: https://github.com/milpflow/milpflow, 2015.

[16] Y. Cai, Y. Yan, Z. Zhang, and Y. Yang, “Survey on converged data center networks with dcb and fcoe: standards and protocols,” IEEE Network, vol. 27, no. 4, pp. 27–32, July 2013.

[17] M. Chen, H. Jin, Y. Wen, and L. V.C.M., “Enabling technologies for future data center networking: a primer,” IEEE Network, vol. 27, no. 4, pp. 8–15, July 2013.

[18] A. Lara, A. Kolasani, and B. Ramamurthy, “Network innovation using openflow: A survey,” Communications Surveys Tutorials, IEEE, vol. 16, no. 1, pp. 493–512, First 2014.

[19] F. Hu, Q. Hao, and K. Bao, “A survey on software defined networking (sdn) and open flow: From concept to implementation,” Communications Surveys Tutorials, IEEE, vol. PP, no. 99, pp. 1–1, 2014.

[20] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, and et al., “Virtual routers as a service: The route flow approach leveraging software defined networks,” in 6th International Conference on Future Internet Technologies 2011 (CFI 11), June 2011.

[21] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, “Automatic configuration of routing control platforms in open flow networks,” SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 491– 492, Aug. 2013.

[22] J. Batalle, J. Ferrer Riera, E. Escalona, and J. Garcia-Espin, “On the implementation of nfv over an open flow infrastructure: Routing function virtualization,” in Future Networks and Services (SDN4FNS), 2013 IEEE SDN for, Nov 2013, pp. 1–6.

[23] S.-Y. Wang, C.-L. Chou and C.-M. Yang, “Estinet open flow network simulator and emulator,” Communications Magazine, IEEE, vol. 51, no. 9, pp. 110–117, September 2013.

[24] ns-3, “Network Simulator 3,” 2014. [Online]. Available: https://www.nsnam.org, 2015.

[25] M. Casado, D. Erickson, I. A. Ganichev, R. Griffith, B. Heller, N. Mckeown, D. Moon, T. Koponen, S. Shenker, and K. Zarifis, “Ripcord: A modular platform for data center networking,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-93, Jun 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-93.html.

[26] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba, “DOT: Distributed OpenFlow Testbed,” in Proceedings of the ACM SIGCOMM 2014 Conference on SIGCOMM, August 2014.

[27] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "MaxiNet: Distributed Emulation of Software-Defined Networks," in IFIP Networking 2014 Conference, 2014.

[28] L. Ghein, MPLS Fundamentals. Cisco Press, 2006.

AUTHOR BIOGRAPHY

Lucio Agostinho Rocha receives your master degree in Computer Network at Federal University of Uberlândia, Minas Gerais, Brazil (2009); doctoral degree in Computer Engineering at State University of Campinas, São Paulo, Brazil (2013); and a post-doctoral degree in Computer Network at Federal University of São Carlos, campus Sorocaba, Brazil (2014). Its publications include researches in distributed systems, robotics, operating research and software defined networking. Currently he is a Computer Science teacher in the Federal Institute of São Paulo, câmpus Araraquara, Brazil (2015).