MILPFlow: a Toolset for Integration of Computational Modelling and Deployment of Data Paths for SDN

Lucio A. Rocha
Department of Computer Science
Federal University of São Carlos
Campus Sorocaba - Sorocaba, Brazil
E-mail: outrosdiasvirao@gmail.com

Fábio L. Verdi
Department of Computer Science
Federal University of São Carlos
Campus Sorocaba - Sorocaba, Brazil
E-mail: verdi@ufscar.br

***Abstract*—Software Defined Networking 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. 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. However, computational modelling for SDN is still few researched in the literature. Computational modelling is the key to describe, evaluate and analyse the most diverse computational problems before its prototyping. In this paper, we propose the integration between computational modelling and the deployment of data paths for SDN. We develop the toolset called MILPFlow to generate computational models of data centers to solve routing problems and to establish data paths between servers according to the solutions of these models. Additionally, data paths are set before data flows are sent through the network. As a consequence, MILPFlow contributes to reduce the overhead to discover network routes among hosts of data centers. We evaluate the effectiveness of our methodology by using Mininet through a set of experiments.**

***Keywords*—SDN, OpenFlow, Network Management.**

I. Introduction

Computational modelling is crucial for network management [1] since it helps to evaluate strategies prior their execution in real scenarios, avoiding to spend resources with problematic situations. In this sense, computational modelling is useful in many aspects, mainly for detecting scenarios where bottlenecks occur before receiving the network data traffic [2].

Computational modelling is a research area related to mathematical modelling of real-world problems, giving solutions to analyze these problems in computers. Additionally, Operational Research is a particular area that groups techniques of mathematical modelling with the objective of defining the best planning for utilization of restricted resources, i.e., it is an optimization process [3]. In this paper, we adopt computational modelling using techniques of the Operational Research to establish data paths for the traffic in SDN networks.

Computational modelling for data centers must take into account the impact on the transport network, and is analyzed by many authors in the literature [4–8]. Many of these studies contemplate modelling of VM placement. The placement of VMs is the problem of selecting the servers where the VMs are allocated in such a way that performance metrics are optimized. These models receive as input only the most relevant features of data center environments, commonly the number of servers, number of VMs, network links, network bandwidth and traffic matrices. These models are adequate to evaluate how the traffic will be carried inside of data centers. Nevertheless, many aspects of real networks, e.g., drop of packets, delay and jitter are difficult to evaluate only with computing modelling. So, it is important to consider how the solution of these models is reflected on real networks.

In this paper, we present MILPFlow (**M**ixed **I**nteger **L**inear **P**rogramming with Open**Flow**)[1], a toolset for the integration of computational modelling and deployment of SDN networks using the OpenFlow protocol. Our approach is two-fold: MILP modelling and the deployment of data paths in the SDN network. We validate our approach using 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 rest of this paper is structured as follows. Section II presents the requirements for integrating computational modelling with the management of OpenFlow rules. Section III presents an evaluation of MILPFlow. Finally, we conclude our paper and contribute an overview of future works in Section IV.

II. Methodology

Figure 1 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.

SDN Topology description: it 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, bandwitdth, delay, and other customized information. In particular, we describe our

[1]MILPFlow is published as open-source software at: https://github.com/milpflow/milpflow.

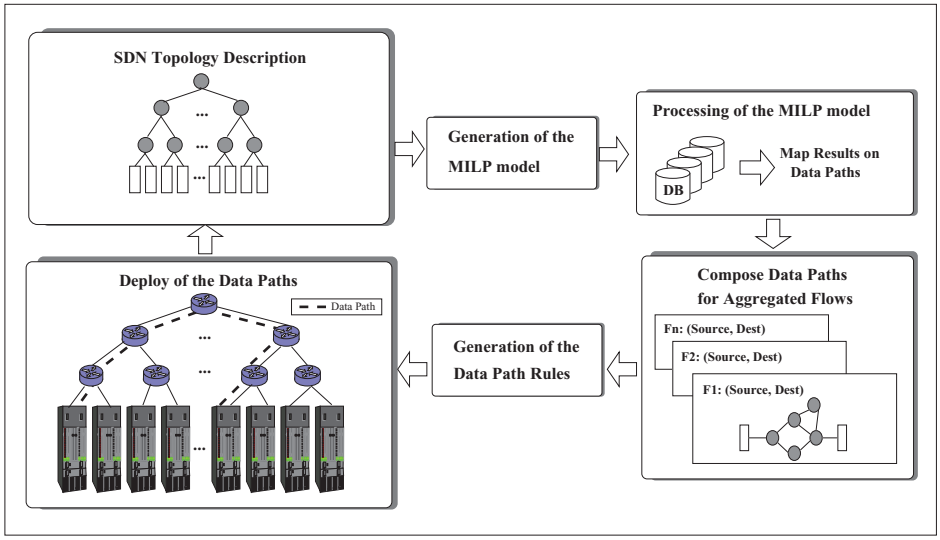


Fig. 1. MILPFlow Methodology.

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.

Generation of the 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.

Processing of the MILP Model: the processing is externally done with MILP solvers, such as CPLEX or Lingo. 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 Figure 2 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 $H1$ and $s2$. Our goal is simplify the mapping of MILP results on data paths.

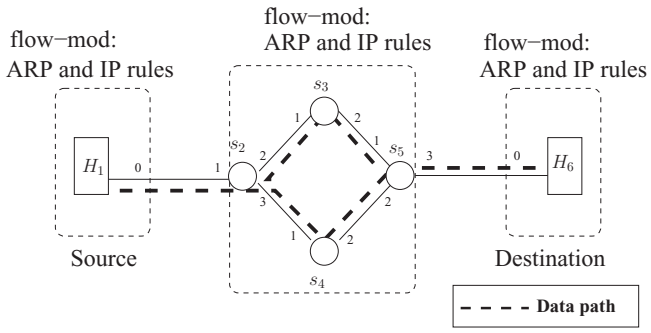


Fig. 2. Mapping of Paths with MILPFlow.

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)

Compose 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 Figure 2. In this example, the sequence $F1[2][3]$ and $F1[2][4]$ are the links between the node $s2$ to reach the nodes $s3$ and $s4$. 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.

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 Figure 2. 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-switch-port 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[2]. As a consequence, it is not necessary to send control

[2]Mininet network parsed from *net* command.

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 to 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 Figure 2. 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: $F1[2][3] = 200$, and between the switches s_2 and s_4: $F1[2][4] = 800$. Then, we set $weight_1$=2, $weight_2$=8, respectively.

Deployment of Data Paths: 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.

III. Experimental Evaluation

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 VirtualBox with XUbuntu 12.04.2 LTS with 1 Gbps NICs, and Open vSwitch compatible-OpenFlow switches. We use Mininet and the Fat-tree topology of Figure 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.

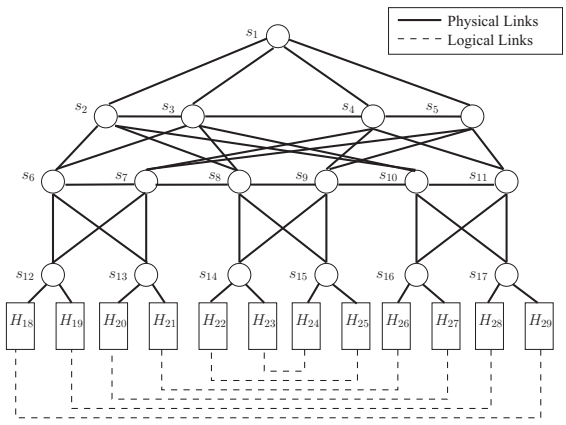
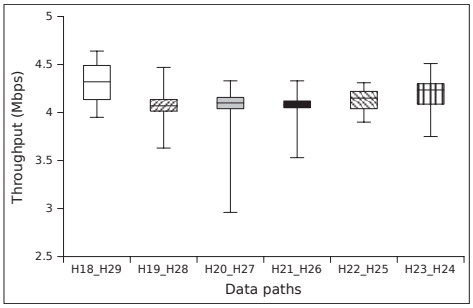
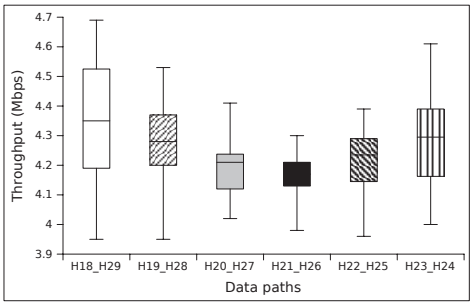


Fig. 3. Fat-tree Topology used in the Experiments.

We use MILPflow to generate the data paths, following the steps of the methodology illustrated in the Figure 1. The SDN topology description is done for the reference topology of the Figure 3. 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 Figure 3. 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 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.



(a) UDP Throughput with MILPFlow.



(b) UDP Throughput with Ryu STP.

Fig. 4. Evaluation of UDP Traffic.

Figure 4 compares MILPFlow versus the STP protocol for UDP traffic. The axis x represents the source and destination TORs by which the traffic is sent. 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.

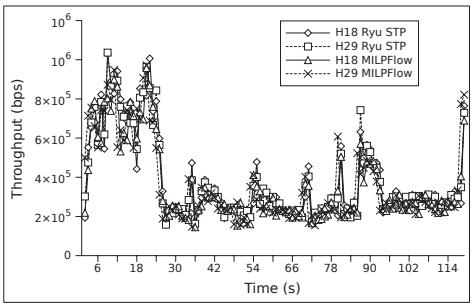
The results with TCP traffic are similar to the UDP traffic since the throughput is quite the same between MILPFlow and STP and the packet losses are bellow 1%. The exact numbers are shown in Table II. The throughput reached by

the STP is a little better (near 90Mbps) when compared to the throughput reached by MILPFlow (near 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.

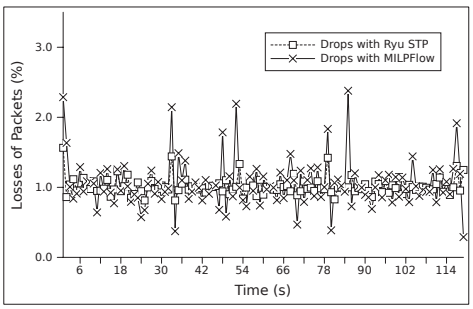
TABLE II. COMPARATIVE EVALUATION WITH IPERF

	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. Figure 5 shows the throughput measurements 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.



(a) Throughput of RTSP Streaming.



(b) Losses of Packets.

Fig. 5. Evaluation of RTSP Throughput.

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. Also, the throughput measurements show higher ponctual peaks in the beginning and in the end of the streaming. We observe that these peaks occur in the parts of the video when more changes of pixels is done. Although the packet losses are not very accurate due to synchronization, the low percentage of losses gives a good indication about 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

IV. CONCLUSIONS AND FUTURE WORKS

This article presents a methodology to integrate computational modelling with management of SDN networks. Our approach aims to promote proactive routing. In order to validate our methodology, we implement the MILPFlow framework, and conduct a set of experiments to evaluate our approach. The main advantage of using MILPFlow is the possibility of doing mathematical modelling together with the deployment of SDN/OpenFlow rules as well as the capability of reconfiguring routes in a finer granularity according to the network administrator needs. 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 modelling jointly with the deployment of rules in the network. Future works are being devoted to evaluate the scalability of MILPFlow for larger topologies.

ACKNOWLEDGMENT

The authors gratefully acknowledge the contribution of CAPES, FAPESP and CNPq, Brazilian funding agencies.

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.

[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] L. Rocha and E. Cardozo, "A Hybrid Optimization Model for Green Cloud Computing," *7th International Conference on Cloud and Utility Computing - UCC 2014*, p. 10, 2014.