Advances in Applied Sciences
2016; 1(2): 37-45
http://www.sciencepublishinggroup.com/j/aas
doi: 10.11648/j.aas.20160102.13

Framework for Traffic Engineering of SDN Data Paths

Lucio Agostinho Rocha

Department of Software Engineering, Federal University of Technology, UTFPR (Universidade Tecnológica Federal do Paraná), Dois Vizinhos, Brazil

Email address:
luciorocha@utfpr.edu.br

To cite this article:
Lucio Agostinho Rocha. Framework for Traffic Engineering of SDN Data Paths. *Advances in Applied Sciences.* Vol. 1, No. 2, 2016, pp. 37-45. doi: 10.11648/j.aas.20160102.13

Received: September 15, 2016; **Accepted**: September 26, 2016; **Published**: October 15, 2016

Abstract: Software Defined Networking (SDN) is an approach to the deployment of future network infrastructures. SDN allows deal with different configurations to a crescent amount of virtualized network devices. In this paper, we offer a framework to support a number of network configurations through computational modeling and deployment of data paths between physical hosts for SDN. Computational modeling is a feasible alternative to measure and analyze the most diverse computational problems before its prototyping. We develop the toolset called Mini-TE (Mini-Traffic Engineering) to perform traffic engineering over computational models of data center topologies, and to set data paths before submission of data streams. As a consequence, Mini-TE contributes to reduce the operating expense to discover routes among hosts of data centers. We want to evaluate the effectiveness of our methodology by using Mininet through a set of experiments.

Keywords: SDN, OpenFlow, Network Management, Network Architecture, Scalability

1. Introduction

The goal of our paper is answer the question: how can we simulate large SDN topologies in a single host? In order to answer this question we conduct this current research.

Software Defined Networking (SDN) is a networking technology, which allows use open protocols to use control software on network devices that typically would use proprietary firmware [1]. With SDN, control and data plane are separated to directly allows control and management in a centralized manner, and the data plane be abstracted more than used with specialized hardware. Despite the fact that management to become directly programmable, SDN architectures are challenges in terms of scalability and support for establishment of complex data paths [2]. Some works [3–6] deal with network scalability defining a logical data plane beyond its physical boundaries. Cloud data centers also benefits from link virtualization due its inherent support to on-demand network provisioning [7]. On the other hands, computational modeling is essential for network management [8] since it helps to evaluate strategies prior their execution, avoiding to spent resources with problematic situations, mainly for detecting bottlenecks before receiving the network data traffic [9]. Computational modeling is a mathematical researching area that presents solutions to analyze non-trivial problems in computers. Additionally, Operational Research is an area that the group's techniques of resource optimization [10].

We argue that computational modeling is important to evaluate large network SDN data paths, although its implementation to be restricted due traditional networks, which are Commodity Off-The-Shelf (COTS) designed by vendors with limited configuration capabilities provided to web administrators.

In fact, there are works [11–13] that deal with network modeling as a Multi-Commodity Flow (MCMF) problem. Much effort has been given to improving scalability and programmability of these networks in data centers [14–17] by using an OpenFlow protocol [18]. This open network protocol allows program the flow-table in different switches and routers. OpenFlow decouples control plane from the data plane, and allows an SDN controller to manage rules to forward traffic. Although be reasonable use well-defined network protocols to automatic discovery of network devices, also is important that these networks be more flexible to current need of network administrator, such as eventual bottlenecks, high demand peaks, fault tolerance, and/or load balance occurrences. Also, Multi Protocol Label Switching (MPLS) is suitable for OpenFlow programmable networks

with changeable data paths [19].

The most common alternative to offer a global view over many physical topology is using a centralized SDN controller [3–6, 20]. Global view is an alternative to turn feasible setups for very large network topologies. But Zerrik et al. [7] claims that these centralized SDN architectures are constructed in way very inflexible. In fact, SDN architectures should be highly flexible with minimal human intervention, and should be scalable and adaptable simultaneous. Even so, one of the major difficulties about implementing decentralized SDN architectures is due the inherent nature of Ethernet ARP broadcast messages to discover new nodes. The increasing of amount of network devices potentially will imply in ARP broadcast storms into network. Another issue is some nontrivial network topologies in reason of some SDN controllers have difficult to solve loops in its topologies [21].

In this paper, in order to deploy large SDN topologies, we set network data paths supported by computational modeling. The main contribution is two-fold: offer an alternative of routing without losses, and an methodology to deploy large virtualized experiments with Mininet. Our approach is directed to use a framework where each physical domain maintains its own OpenFlow rules. This access is distributed because divides the processing of routes through its communicant parts that are in distinct hosts. Finally, our framework combines partial network set-ups from each domain in a single global logical plane. We adopt computational modeling techniques from the Operational Research to establish data paths for the traffic of SDN networks, founded on our previous work [22] with the computational modeling approach.

In this paper we introduce a set-up of data paths with Mininet through separated physical domains. The goal has defined a methodology to deploy scalable virtual links between these domains, in a proactive manner. Our framework, known as Mini-TE (Mini-Traffic Engineering) is used to provide a unified network logical plan to map and deploy large data paths, distributing the nodes and virtual switches in many virtualized hosts running Mininet emulator. Finally, our tests show serious results in concurrent communication for TCP and UDP protocols.

This paper is structured as follows. Section 2 is about related works. Section 3 presents our methodology for integrating computational modeling with the management of OpenFlow rules. Section 4 presents the MiniTE architecture used in the experiments. Section 5 relates the experimental evaluation. Finally, conclusions and future works are done in Section 6.

2. Background

Multi-commodity flow (MCMF) problems are close to multiple demand flows (commodity demands) in network flows with different source and sink nodes [14–17]. These demands are commonly found in large scale networks, such as server farms. In these environments, the integral nature of its communications allows optimization of traffic by traffic engineering (TE) methods. Benson et al. [15] says that existent TE techniques perform 15% at 20% worse than the optimal resolution. Nevertheless, Al-Fares et al. [16] affirm that multirooted trees with many equal-cost paths are the common solution used by many ISPs (Internet Service Providers), but existent multipath protocols may cause substantial bandwidth losses due high collision rates. In fact, our previous work with MPTCP (Multipath TCP) [23] showed that is important includes multi-path TCP as na choice to distribute data traffic via alternative routes. MPTCP is a network protocol designed to forward subflows through disjoint paths. Even using ECMP (Equal Cost MultiPath) many flows may be forwarded by the same data path. MPTCP is not exclusive to OpenFlow networks, but is useful to automatically forward subflows two distinct routes. However, MPTCP will automatically redefine data-paths according specific network requirements, with minor human intervention.

Dynamic traffic engineering techniques for SDN networks has been contemplated by many authors, and applied in data centers. Nevertheless, the applicability of these techniques generally is limited due restrictions of ISP domains in provide detailed information about its infrastructure. A possibility to bypass these restrictions is use emulation/virtualization for experimentation. In this sense, new alternatives TE techniques for large scale environments may be evaluated before prototyping, and provide detailed information about possible leaks, bottlenecks, alternatives routes, fault-tolerance, split/join of data traffic, and many others. SDN leverage these possibilities because a number of experiments may run in short time in controlled domain. Intensive network applications for distribute computing (e.g. MapReduce, peer-to-peer, SOA based applications, and many others) also may be assessed in a few emulated hosts.

A lot of works deal with virtualized experiments using the OpenFlow protocol for SDN networks. Flowvisor [5] is an OpenFlow proxy to enable multiple SDN controllers in a net. The software separates the entire network into slices and one controller can be used to manage each network slice. Multiple controllers can manage the same set of switches for different uses. But Flowvisor is not a SDN controller: it only forwards packets between switches and controllers, and not allow any communication between different controllers what restrict the scalability of data paths setups. An improved distributed FlowVisor [6] address scalability issues to cloud computing environments. Scalable cloud experimentation may be used in OpenStack Neutron with virtual L2 networks [4] and OpenDayLight Virtual Tenant Network and OpenDayLight Virtual Tenant Network (VTN) [3]. VTN and OpenStack Neutron gives a logical abstraction plane of network that separate logical plane from the physical plane, and give possibility to deploy virtual networks without experiencing the entire physical topology.

Although Mininet results are not well accurate for large topologies [24], it is still an affordable free choice to run OpenFlow experiments in a single server. Mininet cluster edition [25] is a working-in-progress prototype to distribute

nodes and connections over a cluster of servers, using resources of each machine to scale the large network. There are some matters about this solution: only one Mininet instance run across the cluster, remote nodes depend of ssh connections to remote Mininet servers, and there is no control about where virtual hosts and virtual switches are physically allocated. On other side, many cloud providers use multi-tenancy packet isolation with virtual switches and conventional packet isolation mechanisms, such as GRE tunnels, MPLS and VLAN. Ahmed et al. [26] proposes the Open virtual Network Management (Open vNMS) to large scale experimentation in clouds. The work is based on elastic L2 isolation. Related at SDN controller, Onix [27] and Hyperflow [28] distribute the control plane but maintain a centralized distributed file system. Kandoo [29] distribute the control plane and act as coordinator between local controllers. Kandoo proposes a hierarquical distribution of controllers in two layers. The bottom layer groups controllers without connection between them. The top layer provides a centralized controller with global network state. However, Zerrik et al. [7] consider that this approach causes some issues: local controllers need that the root controller subscribes itself to each OpenFlow event. This is need to allow the propagation of these events to specific local controllers, i.e., it is necessary to subscribe all OpenFlow events in all local controllers what implies in high amount of information in the current tables of OpenFlow switches. But Kandoos authors not explain how the datapaths are configured.

On other hands, DIFANE [30] is a distributed management architecture that distribute rules to authority switches, and handles wildcards rules according to network dynamics. Many works use frameworks over Mininet as base for SDN experiments. Barford et al. [31] presents a simulation tool known as fs-sdn, but experiments were performed for a little sum of nodes (until 100 nodes). Recent works use frameworks built over Mininet. Murthy et al. [32] use a framework to deploy customs topologies for lesser than 100 clients in a single host running Mininet. Darabseh et al. [33] propose SDStorage as an experimental framework propose SDStorage as an experimental framework built over Mininet to abstract data storage control operations from the storage devices from a centralized controller in the software layer. Minievents [34] is a framework over Mininet that introduces an event generator for the topology, such as TCP and UDP traffic, delay, and bandwidth changes.

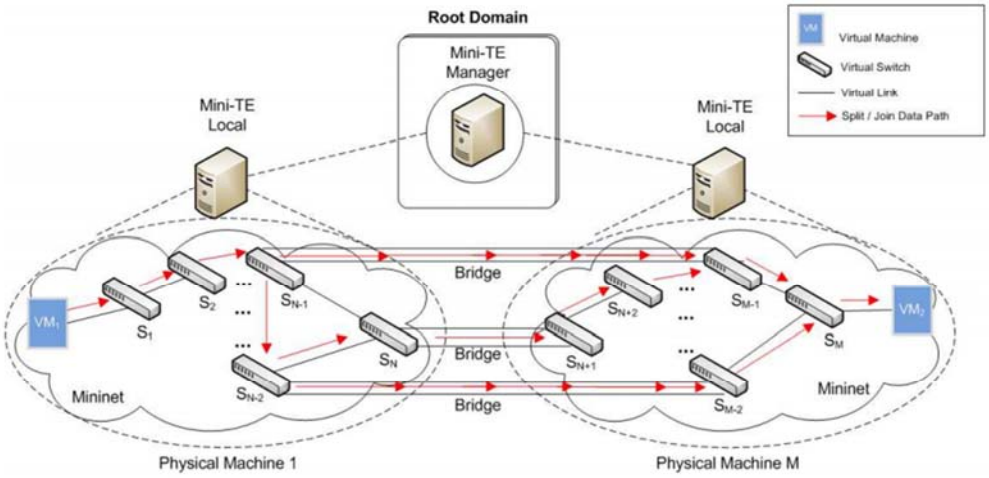

Fig. 1. *Mini-TE Approach to Deploy Data-Paths.*

Other popular software for SDN experiments is Estinet [24]. EstiNet is an OpenFlow network simulator and emulator, and 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 (network-simulator 3). This network simulator uses 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. Wang [35] uses Floodlight controller to compare EstiNet simulator, EstiNet emulator, and Mininet emulator over a set of grid networks of reduced amount of nodes (until 31 nodes). Performance measurement was done with ping of packets to acquire the average RTT among the simulated nodes. EstiNet emulator showed better results about data correctness, RTT performance and scalability.

As an alternative, the toolkit DOT [36] distributes the emulated SDN network using multiple physical machines to provide resources for its network elements. The toolkit uses an embedded algorithm that minimizes the physical bandwidth load and the routine of physical machines need to support the specified network topology. Also, is possible to perform the emulation using DOT inside VMs with Qemu.

Another alternative is use the toolkit Maxinet [37] that span the emulation of larger topologies across physical machines. All switches are checked 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.

3. Methodology

In this section we briefly explain our strategy to deploy data paths by set up flow entries in the switches.

3.1. Mesh Layered Topology

On that point are many network topologies which may be evaluated for large scale virtualized environments from simple linear or balanced trees, or even more complex topologies, such as Hypercube [38], Fat-tree [39], VL2 [40], BCube [41], Portland [42], and many other complex models [43]. These latter topologies have interesting properties which makes them suitable for large scale experiments, such as number of links per node (degree of a node), a small internode distance (diameter), and many alternative paths between pairs for fault tolerance [38]. However, our aim is to evaluate a topology which is bandwidth-intensive and support many alternative data paths inside and outside its layer domain. Although there are various different topologies, we are interested in evaluating the OpenFlow potential to redistribute traffic, independent of the chosen topology.

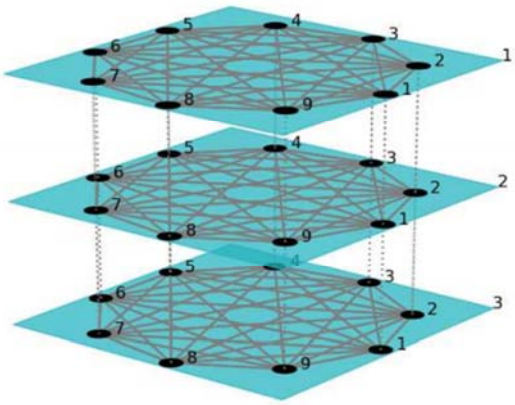

Fig. 2. *Mesh Layered Topology.*

In this sense, we use a mesh layered topology model, as illustrated in Figure 2. Mesh layered topology is a 3-dimensional graph where the nodes are the switches and the edges are the connections between them. Each node keeps links with its neighbor nodes. This scheme is a paradigm where commodity switches are utilized to establish direct links between them. As a result, each switch is able to forward data to/from its direct neighborhood, as illustrated in Figure 2. Our topology model defines:

- Li: is the layer index i;
- si, j: is the virtual switch node j in layer i | i, j ∈ N.

A layer Li is the physical host i with many virtual switch nodes. Each switch node si,j keeps connection with its neighbour switch node, i.e., exist a link between si,j with si+1,j and si−1,j. As a result, we preserve the clear characteristics of mesh topologies whilst extend its model in a well-defined scalable structure. So, our computational modeling will define the routing data path through virtual switches in different layers, although each Mininet domain may specify its own routing protocol.

In our solution, we have control about where virtual hosts and virtual switches are physically instantiated because each domain has full control of its own Mininet instance. Other Works as Maxinet [37], DOT [36], Flowvisor [5], [6], Mininet cluster edition [25] and OpenDayLight VTN [3] not currently give possibility to keep different Mininet instances separated, routing between worker nodes is performed in a transparent way, and distribution of virtual switches is done automatically.

We utilize a lightweight solution by set GRE (Generic Routing Encapsulation) tunnels across physical domains with Open switch (OVS). This permits that each virtual switch communicates itself with its neighbors. OVS is an implementation of emulated switch in the Linux kernel with OpenFlow support, and it is able to create emulated virtual switches, connect them at other virtual and/or physical interface, and use SDN controller to configure its switches. Our deployment of OpenFlow rules is managed entirely via dpctl commands without an SDN controller. Figure 1 illustrates our methodology. Mini-TE Manager component will receive partial topology configuration and pass out over its managed Mini-TE Local components. We explain this process as follows.

3.2. Flow Set-up

Our evaluation depends on set up of flows provided by our linear programming example. Flow entries act to match flows and specify the action for incoming and outcome packets. The action specified by flow entries can be mapped to a group action. The purpose of a group action is define more specific forwarding action to process these incoming packets. A group action contains a bucket, and this latter contains a list of actions. There are four types of groups that are all, select, indirect and fast-failover. We use select group from OF13SoftSwitch to stochastic switching of data traffic. We set a weight to entry data traffic in a round robin model to split incoming traffic for each bucket, e.g., an entry data flow of 1000 units splited in 200 and 800 units will be splited to buckets with weight=2 and weight=8, respectively. The precision by which a packet distribution will be done is undefined. Also, the bucket sharing is determined by the individual bucket's weight divided by the sum of the bucket weights in the group. As a result, we can generate splits and/or join of data traffic. Group tables are available since OpenFlow 1.1 specification, we use group table from OpenFlow 1.5 supported by OVS 2.5.0 specification.

- *Packet in*: it is the message from switch when a packet arrives, and no matching flow entry is found. This message contains the headers and payload of the received packet, and is transmitted to the controller. The Packet in message contains the buffer id where packet is buffered in the switch to aim the controller to sent proper message response to forward the packet in its specific buffer located in the switch.
- *Packet out*: it is the message from controller to forward the received packet to a specific switch port.
- *Flow mod*: it is the message from the controller to a switch in order to create flow entries inside the OpenFlow switch. This message informs how deal with new received packets, rules and actions to be performed.
- *Tunneling*: when a packet need to be forwarded to outside from local Mininet emulated domain, a tunnel is created to bypass its current area via bridges. Tunneling is supported by current Mininet version, but is setup outside its emulated network via MiniTE Local service. An option key id is used to specify many concurrent tunnels in a same virtual ovs switch.

When a package 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 make a packet in event in the controller. Then, the controller will require 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 transported back to the switches to set their flow tables. We illustrate this process in Figure 3. The establishment of these data paths is made out with proactive or reactive methods.

Proactive routing methods in SDN networks, assure that paths are set before packets are sent on the mesh. The advantage is that they do not generate extra network overhead and do not affect the network performance to discover its endhosts. When packets arrive on 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 on these topology, which is not always feasible for all networks.

Reactive routing methods occur when new a packet hits a switch without a matching in the flow entry. Then, such packet must go to the controller, which in turn uses control messages to automatically discover its end-hosts and obtain the active links before the data traffic to be forwarded in the net. 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 determine 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 bring in dynamic traffic engineering the paths in SDN networks. Proactive methods require syncronization between the controller device that deploy paths. But reactive methods introduce additional overhead in large networks and influence the network functioning. So, we adopt a proactive method to establish large data paths between physical hosts.

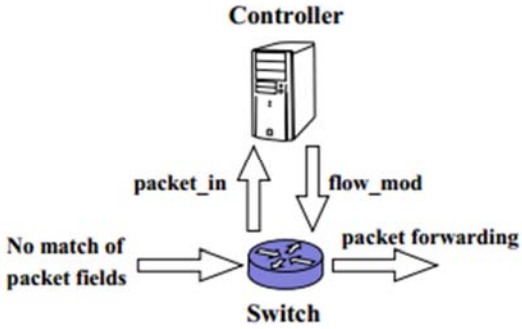

Fig. 3. *OpenFlow Packet Forwarding.*

3.3. Processing of the MILP Model

The processing is done with Mini-TE Local component which have MILP solvers, such as CPLEX or Lingo. The result obtained from solving this MILP model is a set of paths to forward data between virtual hosts, without over-utilization of the network links. Each Mini-TE Local component of our architecture use our previous work MILPFlow [22] to generate data-paths to its own domain. Then, Mini-TE Manager receives these results to get an abstract network's global view.

We present an example in Figure 4 and its corresponding mapping results in Table I. The data structure F1 keeps the whole information about the links with aggregated-flows F1.

So, F1 [1] [1] [2] = 1000 indicates that *Layer1* has 1000 units of the flow F1 goes through the link between the nodes H1 and s2;

Also, F1 [2] [8] [10] = 1000 is *Layer2* who has 1000 units of the flow F1 that goes through the link between the nodes s8 and h10.

Table 1. *Mapping of ports with Milpflow.*

MILPFlow results	Mininet Topology in Layer	Mapping of Ports
F1 [1] [1] [2]=1000	(h1,s2)	(h1,0,s2,1)
F1 [1] [2] [3]=0	(s2,s3)	(s2,2,s3,1)
F1 [1] [2] [5]=0	(s2,s5)	(s2,3,s5,1)
F1 [1] [4] [5]=0	(s4,s5)	(s4,2,s5,2)
F1 [2] [6] [7]=800	(s6,s7)	(s6,2,s7,1)
F1 [2] [7] [8]=800	(s7,s8)	(s7,2,s8,1)
F1 [2] [6] [9]=200	(s6,s9)	(s6,3,s9,1)
F1 [2] [9] [8]=200	(s9,s8)	(s9,2,s8,2)
F1 [2] [8] [10]=1000	(s8,h10)	(s8,3,h10,0)

3.4. Compose Data Paths

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 virtual host to reaches its destination virtual host. For this we use an algorithm similar at the depth-first search

algorithm in order to explore as far as possible each branch in the path of F1. Figure 4 illustrates this example.

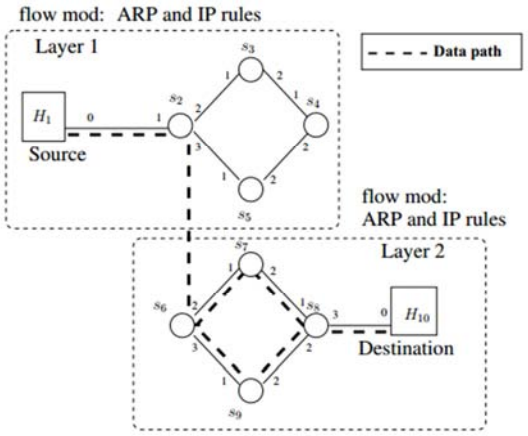

Fig. 4. *Mapping of Layered Paths with Mini-TE Manager.*

In this example, the sequence F1 [2] [6] [7] and F1 [2] [6] [9] are the links of the node s6 to reach the nodes s7 and s9, respectively. So, we have a split condition in this way. Similarly, the sequence F 1 [2] [7] [8] and F 1 [2] [9] [8] are a join condition near at the destination.

3.5. Generation of Data Path Rules

Our approach considers aggregation of many micro-flows in end-to-end data path before submission. We describe this step with an example of a flow generated from host H1 to host H10, as exemplified in Figure 4. We create switches si in crescent order to communicate with each other via communication ports. We combine MILP results with Mininet to create a mapping of MILP ports, as depicted in Table I. The first column in Table I shows the route created by MILPFlow to go from host H1 to host H6. The second column indicates the hosts and switches in the Mininet topology in each layer and ultimately in the last column, we see each host-port and 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 connection that connects switches is necessary to establish connectivity hop by-hop, and this task is generally performed by OpenFlow controller. However, we acquire this data directly from Mininet (parsed from net command). As a consequence, it is not necessary to transmit 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 developers to create custom functions to split flows in the network as easily as their subsequent joins. For each group table, the value of the variable weight indicates the quantity of traffic that should be forwarded through its sub-paths, as occurs in the switches s2 and s5 in Figure 4. We set the normalized weight value of the MILP result. As an example, we map the resultant flow of 800 units in the linkup between the switches s6 and s7: F1 [2] [6] [7] = 800, and between the switches s6 and s9: F 1 [2] [6] [9] = 200. Then, we set weight1=8, weight2=2, respectively. Finally, on each switch with split the group mod rule is set as follows:

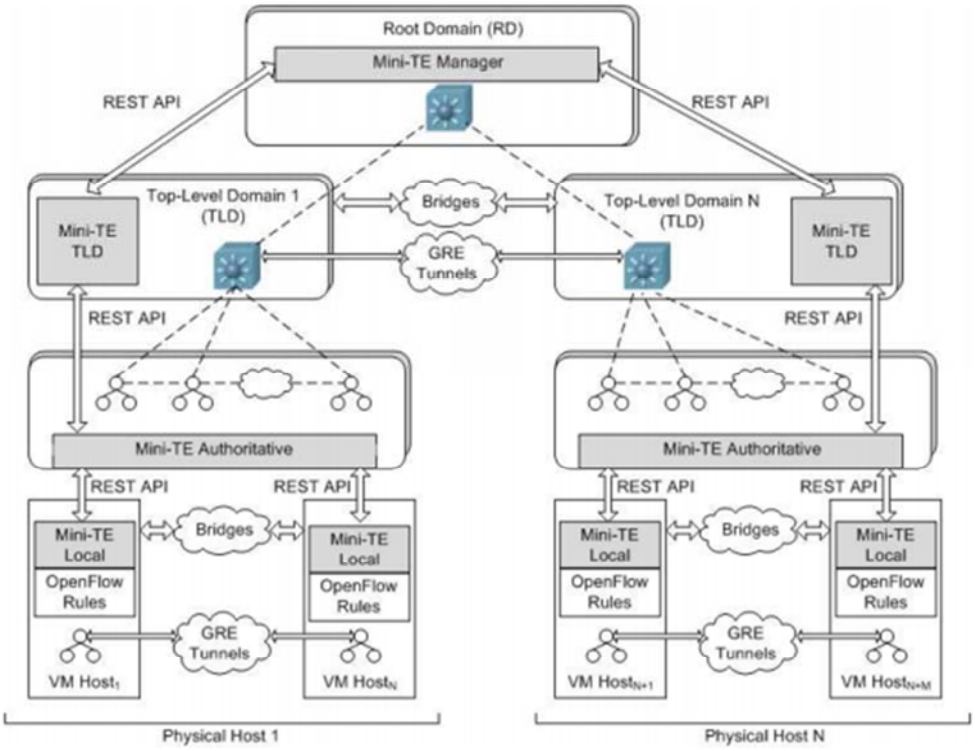

Fig. 5. *Mini-TE Distribute Architecture for Deployment of Data Paths.*

```
dpctl tcp:<HOST_IP>:<HOST_PORT> group-mod\
cmd=add,type=<TYPE>,group=<GROUP_ID>\
weight=<WEIGTH_1>,port=<PORT_1>,\
group=all output=<OUT_PORT_1> \
weight=<WEIGTH_2>,port=<PORT_2>,\
group=all output=<OUT_PORT_2>
```

3.6. Deployment of Data Paths

Mini-TE Local service 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 our REST API. This API maps dpctl commands to set input and output ports in each switch of the data path. Mini-TE Local service creates for each instantiated virtual switch a GRE tunnel with your correspondence vicinity in order to permit forward traffic data.

4. Mini-TE Architecture

Mini-TE Architecture is showed in Figure 5. Mini-TE Local components send its virtual switches and virtual hosts description to Mini-TE Manager. Inside each physical host are instantiated the Mini-TE Local, Mini-TE Authoritative, and Mini-TE TLD components. We use GRE tunnels between physical hosts and between virtual machines because we are in a controlled field. A possible improvement would be using ssh for secure connection between physical servers. A description about each part of our architecture is given as follows:

Mini-TE Manager: it is the daemon which works in the root domain, and receives/sends messages in REST API to forward packets to other physical hosts. Mini-TE manager communicates via REST API with many Mini-TE TLD, or directly with Mini-TE Local if there is a single physical host. Nevertheless, many Mini-TE Managers may communicate each other to expand the scale of virtualized experiments.

Mini-TE TLD: This component is necessary only if we want to communicate with virtual hosts from another physical host. Mini-TE TLD is the daemon which receives/send messages in REST API from Mini-TE Authoritative and send to its registered Mini-TE Manager.

Mini-TE Authoritative: it is the daemon which join many local context in a single global view of topology. This daemon receives/send messages in REST API about the entire global topology view which is operating on a single physical host.

Mini-TE Local: it is the daemon which receives/send messages in REST API to deploy OpenFlow rules to Mininet local context. A same physical host may run many virtual machines, each one running its own Mininet context. So, MiniTE Local daemon must be given on each virtual machine. This component is responsible to make tunnels for each virtual switch with its correspondence in upper and lower layers.

5. Experimental Evaluation

We define a set of methods to reduce the reconfiguration time of network services and minimize the network configuration errors. Our Linux environment was configured with ulimit 65535, VirtualBox NIC virtualized Intel PRO/1000 MT Server, ovsvsctl (Open vSwitch) 2.5.0, DB Schema 7.12.1, dpctl 1.3 from CPqD, and Mininet. Preliminary evaluations showed that instantiated GRE tunnels in Mininet didn't make significative impact on execution.

The required time to compute data path rules will depend on topology, size, but we don't consider this matter as a limiting factor because each physical domain will be able to process its own MILP model. Also, the required time to setup data path rules is not a restriction because will be dependent of the physical machine resources. We want measure a set of scenarios on an Intel Pentium 4 with 3 GHz, 1GB of RAM for Mini-TE Local, and MiniTE Manager on a Core 2 Duo with 2.4GHz, 2GB of RAM. Our experiments run inside VMs of VirtualBox with XUbuntu 14.04.2 LTS with 1 Gbps NICs, and Open vSwitch compatible OpenFlow switches. We use two layers of the Mesh Layered topology with 10 hosts for each one of 10 switches (100 hosts in each layer). Our experiments will be done with Iperf software, and streaming of video using the VLC application with Real-time Streaming Protocol (RTSP).

The purpose of this evaluation is to show the feasibility of the tool in terms of mathematical modeling and the deployment of flows. We use MILPflow to generate the data paths, following the steps of the methodology. For this experiment we set aggregated flows of 1000 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 will be done taking into account UDP and TCP traffic. In both experiments we want to run Iperf measurements 30 times for each pair of virtual hosts, i.e., h1 from Layer1 with h11 of Layer2, h2 from Layer1 to h12 of Layer2, and so on. Also, we want to collect data every 5 seconds for each measurement. For UDP traffic we want to use Iperf to submit 5Mbps between the virtual hosts. For TCP traffic we want to run similarly, but without restriction about the amount of data to be forwarded among the virtual hosts.

Our preliminaries results show that mesh topologies are useful alternatives to evaluate large scale deployment of data paths. However, our current infrastructure is not able to perform the current tests of Mini-TE architecture. The reason is that our physical hosts not support more than 50 virtualized hosts. So, we are considering the evaluating as a work-in-progress while we can't run these tests in our own physical infrastructure. Whole software developed until this moment is available on-line at https://github.com/mini-te/mini-te.

We expect that the results with TCP traffic be similar to the

UDP traffic since the MILP generates data paths without losses. Also, MILPFlow modeling is important 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.

6. Conclusions and Future Works

Our article presents a methodology to integrate computational modeling with management of SDN networks in order to deploy very large SD topologies.

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 modeling 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 modeling jointly with the deployment of rules in the network.

Our approach not obligate that vendors modify its current network hardware, and no kernel modification is need to run changeable data paths, as must be done with MPTCP. We consider in this paper that control programs installing an end-to-end path on a per-flow basis is not scalable due limited switch memory. However, data-paths can be established for aggregate rules matching a large number of micro-flows. As a consequence, the information propagation delay in SDN would be similar at traditional networks. Our experiments with Mininet and OpenFlow demonstrate that proactivelly deploy rules on switches is an affordable alternative for aggregation of data paths, eliminating major of control requests in the control plane. However, the cost is loss of precision and reactivity in the controller in fault events, bottlenecks, high peaks of demand, and others. These problems are not caused fundamentally by SDN, but can be addressed without losing the benefits of SDN. The level of flexibility to accommodate network programming and management at scale are challenges, even for traditional networks.

We note that SDN/OpenFlow experiments with computational modeling are still few present in the literature. 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. As future work, we would like to extend our approach to mobile SDN networks, and simplify the setup with automated packages.

Acknowledgment

The author gratefully acknowledges the contribution of the Grupo de Pesquisa em Engenharia de Software e Informática (GPESI) - (http://dgp.cnpq.br/dgp/espelhogrupo/6364090264037055) a Brazilian research group.

References

[1] M. K. Shin, K. H. Nam, and H. J. Kim, “Software-defined networking (SDN): A reference architecture and open APIs,” in 2012 International Conference on ICT Convergence (ICTC), 2012.

[2] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, “On scalability of software-defined networking,” IEEE Communications Magazine, 2013.

[3] “OpenDayLight User Guide: Virtual Tenant Network,” 2016, p. 62. [Online]. Available: https://www.opendaylight.org/sites/opendaylight/files/bk-user-guide.pdf

[4] “Neutron - OpenStack,” 2016. [Online]. Available: https://wiki.openstack.org/wiki/Neutron

[5] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “FlowVisor: A Network Virtualization Layer,” in Deutsche Telekom Inc. R&D Lab, Stanford University, Nicira Networks, Technical Report, 2009.

[6] L. Liao, A. Shami, and V. C. M. Leung, “Distributed FlowVisor: a distributed FlowVisor platform for quality of service aware cloud network virtualisation,” IET Networks, vol. 4, no. 5, pp. 270–277, 2015.

[7] S. Zerrik, A. E. ouadghiri, D. E. ouadghiri, R. Atay, M. Bakhouya, and J. Gaber, “Towards a decentralized and adaptive software-defined networking architecture,” in Next Generation Networks and Services (NGNS), 2014 Fifth International Conference on, May 2014.

[8] 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.

[9] 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.

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

[11] M. Moshref, M. Yu, A. Sharma, and R. Govindan, “vcrib: Virtualized rule management in the cloud,” in USENIX HotCloud, USA, 2012.

[12] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: scaling flow management for highperformance networks,” in SIGCOMM CCR, 2011.

[13] N. Kang, Z. Liu, J. Rexford, and D. Walker, “Optimizing the One Big Switch Abstraction in Software Defined Networks,” in CoNEXT, ACM, 2013.

[14] 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, 2010.

[15] 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, 2011.

[16] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow scheduling for data center networks," in 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10, San Jose, USA, 2010.

[17] Y. Li and D. Pan, "OpenFlow based load balancing for Fat-Tree networks with multipath support," in 12th International Conference on Communications (ICC 13), Budapest, Hungary.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," in ACM SIGCOMM, 2008.

[19] A. R. Sharafat., S. Das, G. Parulkar, and N. Mckeown., "MPLS-TE and MPLS VPNs with OpenFlow," in ACM SIGCOMM, 2011.

[20] M. Erel, E. Teoman, Y. Ozcevik, G. Secinti, and B. Canberk, "Scalability analysis and flow admission control in mininet-based SDN environment," in Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on, Nov 2015, pp. 18–19.

[21] POX Wiki, 2015. [Online]. Available: https://openflow.stanford.edu/display/ONL/Pox+Wiki

[22] L. A. Rocha and F. L. Verdi, "MILPFlow: A toolset for integration of computational modeling and deployment of data paths for SDN," in 2015 IFIP/IEEE International Symposium on Integrated Network Management, 2015.

[23] M. Sandri, A. Silva, L. A. Rocha, and F. L. Verdi, "On the Benefits of Using Multipath TCP and OpenFlow in Shared Bottlenecks," in 2015 IEEE 29th International Conference on Advanced Information Networking and Applications, 2015.

[24] S. Y. Wang, C. L. Chou, and C. M. Yang, "Estinet openflow network simulator and emulator," IEEE Communications Magazine, vol. 51, no. 9, pp. 110–117, September 2013.

[25] C. Burkard, "Mininet Cluster Edition," 2014. [Online]. Available: https://github.com/mininet/mininet/wiki/Cluster-Edition-Prototype

[26] M. F. Ahmed, C. Talhi, M. Pourzandi, and M. Cheriet, "A SoftwareDefined Scalable and Autonomous Architecture for Multi-tenancy," in Cloud Engineering (IC2E), 2014 IEEE International Conference on, March 2014, pp. 568–573.

[27] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in In Proc. OSDI, 2010.

[28] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863133.1863136

[29] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ser. HotSDN '12. New York, NY, USA: ACM, 2012, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342446

[30] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-Based Networking with DIFANE," SIGCOMM'10, 2010.

[31] M. Gupta, J. Sommers, and P. Barford, "Fast, accurate simulation for sdn prototyping," in Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ser. HotSDN'13. New York, NY, USA: ACM, 2013, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491202

[32] C. Pal, S. Veena, R. P. Rustagi, and K. N. B. Murthy, "Implementation of simplified custom topology framework in mininet," in Computer Aided System Engineering (APCASE), 2014 Asia-Pacific Conference on, Feb 2014, pp. 48–53.

[33] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "Sdstorage: A software defined storage experimental framework," in Cloud Engineering (IC2E), 2015 IEEE International Conference on, March 2015, pp. 341–346.

[34] C. Giraldo, "Minievents: A mininet Framework to define events in mininet networks," 2015. [Online].

[35] S. Y. Wang, "Comparison of sdn openflow network simulator and emulators: Estinet vs. mininet," in 2014 IEEE Symposium on Computers and Communications (ISCC), June 2014, pp. 1–6.

[36] 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.

[37] 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.

[38] L. N. Bhuyan and D. P. Agrawal, "Generalized Hypercube and Hyperbus Structures for a Computer Network," in IEEE Transactions on Computers, vol. c-33, no. 4, April 1984.

[39] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in SIGCOMM Comput. Commun. Rev., vol. 38, Aug. 2008.

[40] A. Greenberg, J. R. Hamilton, N. Jain, and et al., "VL2: a scalable and flexible data center network," in Commun. ACM, vol. 54, March 2011.

[41] C. Guo, G. Lu, D. Li, and et al., "Bcube: a high performance, servercentric network architecture for modular data center," in Proceedings of the ACM SIGCOMM 2009 Conference on Data communication, SIGCOMM'09, New York, 2009.

[42] R. N. Mysore, A. Pamboris, N. Farrington, and et al., "PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric," in ACM SIGCOMM.

[43] P. J. Mucha, T. Richardson, K. Macon, and et al., "Community Structure in Time-Dependent, Multiscale, and Multiplex Networks," in Science AAAS - American Association for the Advancement of Science, vol. 328, no. 5980.